Feed-Forward Neural Networks and Genetic Algorithms for Automated Financial Time Series Modelling

Jason Kingdon

Thesis Submitted for the degree of
Doctor of Philosophy
of the University of London.

October 1995

Department of Computer Science
University College London

ACKNOWLEDGEMENTS

I would like to thank all my colleagues in the Computer Science Department at UCL for their friendship and advice. In particular, I would like to thank Laura Dekker and Dr Suran Goonatilake for helpful comments and suggestions on earlier drafts of this work. I would also like to thank members of the Intelligent Systems Lab for joint work and collaborations that we have undertaken during my time at UCL. In particular my thanks go to Dr Ugur Bilge, Konrad Feldman, Dr Sukhdev Khebbal, Anoop Mangat, Dr Mike Recce, Dr Jose Ribeiro and John Taylor. Finally I would also like to thank my supervisors Dr. Derek Long and Professor Philip Treleaven.

Abstract

This thesis presents an automated system for financial time series modelling. Formal and applied methods are investigated for combining feed-forward Neural Networks and Genetic Algorithms (GAs) into a single adaptive/learning system for automated time series forecasting. Four important research contributions arise from this investigation: i) novel forms of GAs are introduced which are designed to counter the representational bias associated with the conventional Holland GA, ii) an experimental methodology for validating neural network architecture design strategies is introduced, iii) a new method for network pruning is introduced, and iv) an automated method for inferring network complexity for a given learning task is devised. These methods provide a general-purpose applied methodology for developing neural network applications and are tested in the construction of an automated system for financial time series modelling. Traditional economic theory has held that financial price series are random. The lack of *a priori* models on which to base a computational solution for financial modelling provides one of the hardest tests of adaptive system technology. It is shown that the system developed in this thesis isolates a deterministic signal within a Gilt Futures prices series, to a confidences level of over 99%, yielding a prediction accuracy of over 60% on a single run of 1000 out-of-sample experiments.

An important research issue in the use of feed-forward neural networks is the problems associated with parameterisation so as to ensure good generalisation. This thesis conducts a detailed examination of this issue. A novel demonstration of a network's ability to act as a universal functional approximator for finite data sets is given. This supplies an explicit formula for setting a network's architecture and weights in order to map a finite data set to arbitrary precision. It is shown that a network's ability to generalise is extremely sensitive to many parameter choices and that unless careful safeguards are included in the experimental procedure over-fitting can occur. This thesis concentrates on developing automated techniques so as to tackle these problems.

Techniques for using GAs to parameterise neural networks are examined. It is shown that the relationship between the fitness function, the GA operators and the choice of encoding are all instrumental in determining the likely success of the GA search. To address this issue a new style of GA is introduced which uses multiple encodings in the course of a run. These are shown to out-perform the Holland GA on a range of standard test functions. Despite this innovation it is argued that the direct use of GAs to neural network parameterisation runs the risk of compounding the network sensitivity issue. Moreover, in the absence of a precise formulation of *generalisation* a less direct use of GAs to network parameterisation is examined. Specifically a technique, *artificial network generation* (ANG), is introduced in which a GA is used to artificially generate test learning problems for neural networks that have known network solutions. ANG provides a means for directly testing i) a neural net architecture, ii) a neural net training process, and iii) a neural net validation procedure, against generalisation. ANG is used to provide statistical evidence in favour of Occam's Razor as a neural network design principle. A new method for pruning and inferring network complexity for a given learning problem is introduced. *Network Regression Pruning* (NRP) is a network pruning method that attempts to derive an optimal network architecture by starting from what is considered an overly large network. NRP differs radically from conventional pruning methods in that it attempts to hold a trained network's mapping fixed as pruning proceeds. NRP is shown to be extremely successful at isolating optimal network architectures on a range of test problems generated using ANG. Finally, NRP and techniques validated using ANG are combined to implement an Automated Neural network Time series Analysis System (ANTAS). ANTAS is applied to the gilt futures price series The Long Gilt Futures Contract (LGFC).

Table of Contents

Chapter 1: Learning Systems and Financial Modelling

1.1 Introduction 1

1.2 Adaptive Systems and Financial Modelling 3

1.2.1 Financial Modelling: The Efficient Markets Hypothesis 3
1.2.2 Learning Systems 4
1.2.3 Technical Issues 5

1.3 Time Series Analysis 7

1.3.1 Fundamentals of Time Series Forecasting and Learning 8

1.4 A Brief History of Neural Networks 9

1.4.1 The Development of Neural Net Techniques 11
1.4.2 More Recent Issues 12

1.5 Thesis Overview 13

1.5.1 Research Objectives 13
1.5.2 Thesis Structure 13
1.5.3 Research Contribution 14

1.6 Summary 16

Chapter 2: Adaptive Systems and Financial Modelling

2.1 Financial Modelling 17

2.2 The Problems with Financial Modelling 18

2.2.1 Fuzzy Rationality and Uncertainty 19
2.2.2 Efficient Markets and Price Movement 20

2.3 Evidence Against the Efficiency Hypothesis 21

2.4 An Adaptive Systems Approach 22

2.5 Neural Nets and Financial Modelling 24

2.5.1 Comparisons between Neural Nets and other Time Series methods 26

2.6 Genetic Algorithms in Finance 28

2.6.1 The Genetic Algorithm Search Technique 28
2.6.2 Applications of Genetic Algorithms 31

2.7 Summary **31**

Chapter 3: Feed-Forward Neural Network Modelling

3.1 Neural Net Search **32**

3.2 MLP Training: The Model **33**

3.3 MLP: Model Parameters **35**

3.4 The Data **35**

3.5 MLP: Training Parameters **37**

3.5.1 Architecture 37
3.5.2 Activation Function 39
3.5.3 Learning Rules, Batch and On-Line Training 40

3.6 Network Performance **41**

3.6.1 Convergence 41
3.6.2 Network Validation and Generalisation 43
3.6.3 Automated Validation 45

3.7 Summary **46**

Chapter 4: Genetic Algorithms

4.1 Using Genetic Algorithms **47**

4.2 Search Algorithms **48**

4.2.1 The GA Search Process: The Simple GA. 49
4.2.2 Schema Analysis 50
4.2.3 Building Blocks Under Review 52

4.3 GA Parameters **52**

4.3.1 The Shape of Space 53
4.3.2 Population Encodings 57
4.3.3 Crossover, Selection, Mutation and Populations 59

4.4 A Strategy for GA Search: Transmutation **63**

4.4.1 Five New Algorithms: Multi-Representation GAs (MR-GAs) 65

4.5 Summary **68**

Chapter 5: Hypothesising Neural Nets

5.1 System Objectives 71

5.2 Hypothesising Neural Network Models 72

5.3 Occam's Razor and Network Architecture 72

5.3.1 Existing Regulisation and Pruning Methods 73
5.3.2 Why use Occam's Razor? 73

5.4 Testing Occam's Razor 74

5.4.1. Generating Time Series 74
5.4.2. Artificial Network Generation (ANG) 75
5.4.3 ANG Results 75
5.4.4 Testing Architectures 77

5.5 Design Strategies using Occam's Razor 78

5.5.1. Minimally Descriptive Nets 79
5.5.2 Network Model 80
5.5.3 Network Regression Pruning (NRP) 81
5.5.4 Results of NRP on ANG Series 83
5.5.5 Interpretation of the Pruning Error Profiles 85
5.5.6 Determining Topologies 87

5.6 Validation 88

5.7 GA-NN Hybrids: Representations 89

5.7.1 Fitness Measures for GA-NN Hybrids 90
5.7.2 Neural Networks and GAs: Fitness Measure for Generalisation 90

5.8 Summary 91

Chapter 6: Automating Neural Net Time Series Analysis

6.1 System Objectives 92

6.2 ANTAS 93

6.3 Primary Modelling 96

6.3.1 Automating the use of Neural Nets 96
6.3.2. GA Rule Based Modelling 98

6.4 Secondary Modelling 99

6.4.1 Generating Secondary Models 99

6.4.2 Model Integration 100
6.4.3 Model Performance Statistics 101

6.5 Validation Modules 102

6.6 Control Flow 103

6.6.1 Neural Net Control 103
6.6.2 GA Control 105

6.7 Summary 106

Chapter 7: The Data: The Long Gilt Futures Contract

7.1 The Long Gilt Futures Contract 107

7.2 The LGFC Data 108

7.2 1 Time series Construction 108

7.3 Secondary Data 110

7.4 Data Preparation 111

7.4.1 LGFC Data Treatment 111
7.4.2 Using Moving Averages 113

7.5 Data Treatment Modules 114

7.5.1 Moving Average Modules 114

7.6 Efficient Market Hypothesis and the LGFC 115

Chapter 8: Experimental Results

8.1 Experimental Design 117

8.2 Phase I - Primary Models 118

8.2 1 NN Hypothesis Modules (Phase I) 118
8.2.2 Results for GA-NN Module 119
8.2 3 In-Sample Testing and Validation of the 15-4 Neural Network 123

8.3 GA-RB Module and Combined Validation 124

8.4 Phase II - Secondary GA-RB Models 128

8.4.1 Secondary Model Control Module 129

8.5 Phase III - Validation and Simulated Live Trading 130

8.6 Controls: Analysis of ANTAS 135

8.6.1 Choosing a Network Architecture 135
8.6.2 GA Control Tests 136
8.6.3 Second Order Modelling 136

8.7 ANTAS: Conclusions 137

8.8 Summary 138

Chapter 9: Summary Conclusions and Future Work

9.1 Thesis Motivations 139

9.2 Objectives: Neural Networks and Learning 139

9.3 Thesis Outline and Research Contribution 140

9.3.1 Multiple Representation Genetic Algorithms using Base Changes 141
9.3.2 Artificial Network Generation 143
9.3.3 Network Regression Pruning 144
9.3.4 ANTAS and the Long Gilt Futures Contract 145
9.3.5 Results 146

9.4 Conclusions 148

9.5 Future Work 149

Appendix:

References 153

Appendix A: A1

Appendix B: B1

Appendix C: C1

List of Tables and Figures

Figure 1.4.1: A Conventional Node 10

Figure 1.4.2: Four-Layer Neural Network 11

Table 2.2.1: Versions of the Efficient Market Hypothesis 20

Table 2.5.1: Neural network forecasting comparisons 27

Figure 2.6.1: Genetic Algorithm Evolutionary Cycle 28

Figure 2.6.2: Genetic Operators a) Crossover, b) Mutation 30

Figure 3.2.1: Network Time Series Training. 34

Table 3.3.1: MLP training parameters 35

Figure: 3.4.1: Network Forecast on the Symmetric Ramp 36

Figure 3.6.1: Network Training Mean Square Error Profiles. 42

Figure 3.6.2: Network Forecast for a Random Series (Lottery Numbers). 42

Table 4.3.1: The function DF1 using standard binary encoding. 54

Figure 4.3.1: Binary cube and Base 3 hypercube. 55

Table 4.3.2: Permuted binary encoding for the function DF1 55

Table 4.3.3: Base 3 representation of DF1. 55

Figure 4.3.2: DF1 convergence using 3-types of GA encoding. 56

Table 4.3.4: The function DF2 56

Figure 4.3.3: Base 2 and Base 5 encodings on DF2. Population size of 20. 57

Figure 4.3.4: Simple fitness function F1. 60

Table 4.3.5: Example population for function F2. 61

Table 4.3.6: The sampling probabilities for Table 4.3.5 62

Figure 4.4.1: Multi-representations with communication via migration 66

Table 4.4.1: GAs using multi-representations on fitness functions 68

Figure 5.4.1: ANG generated series. 76

Figure 5.4.2: Networks trained on the 12-7-1 ANG generated series. 76

Figure 5.4.3: MSE of iterated forecasts for 10-6-Series and 14-7-Series. 77

Table 5.4.1: Ten neural network architectures trained on five ANG series 78

Figure 5.5.1: Iterated forecast by 20-10-1 network for 16-9-Series. 83

Figure 5.5.2: NRP applied to the ANG generated test series. 84

Figure 5.5.3: NRP applied to the 14-7-ANG generated series. 85

Table 5.5.1: Complexity Identify used for the 5 ANG series using Eq. 5.7 86

Table 5.5.2: Complexity Identifier (Eqn. 5.8) used for the 5 ANG series. 87

Table 5.5.3: Suggested complexities for Network Architectures for Experiment 1. 88

Figure 6.2.1: ANTAS - Functional Lay-Out. 94

Figure 6.2.2: ANTAS - Modular Lay-Out 95

Figure 6.4.1: Neural Network performance file. 102

Figure 6.6.1: Neural Net control file. 105

Figure 6.6.2: Genetic Algorithm Control File. 106

Figure 7.2.1: LGFC price data from LIFFE. 109

Figure 7.2.2: The LGFC constructed price series (18/11/82 - 27/8/85). 109

Figure 7.2.3: The LGFC Price series. 110

Figure 7.2.4: The LGFC adjusted price series. 110

Figure 7.3.1: The Gilt Indices as compared to the LGFC for the training data. 111

Figure 7.4.1.: Forecast accuracy for a 34-day moving average and raw LGFC. 112

Figure 7.4.2: The moving average and raw LGFC series. 114

Figure 7.5.1: LGFC moving average and raw series price movements (18/11/82 to 27/8/85).116

Table 7.5.1: Moving Averages Forecast Horizons for the LGFC (18/11/82 to 27/8/85). 116

Figure 8.2.1: NRP applied to a 20-10-1 network trained on 250 days of the LGFC. 120

Table 8.2.1: The hypothesised complexities for a MLPs used for the LGFC. 120

Figure 8.2.2: Three typical GAs applied to multi-evaluation time series fitness 121

Table 8.2.2: Neural Network Configuration produced via the GA 122

Figure 8.2.3: Iterated forecast from NN-1 produced by GA (15-4-1 Net). 122

Figure 8.2.4: Corrupted In-Sample Recall from NN-1. 122

Figure 8.2.5: In-Sample Recall for NN-1 for GA-In-Sample training set 123

Table 8.2.3: Neural Net configurations inferred by GA design process. 123

Table 8.2.4: Neural Networks tested over 200 day contiguous LGFC price series 124

Figure 8.3.1: Histogram of correctly forecasted price direction for the LGFC 125

Figure 8.3.2: Histogram of incorrectly forecasted price direction for the LGFC 126

Figure 8.3.3: Histogram of correct forecasted price direction for the LGFC 126

Figure 8.3.4: Histogram of probability of correct LGFC price trend forecast, ... 127

Table 8.3.1: Data produced via Neural Network Validation Module. 127

Figure 8.3.5: Histogram of probability of correct LGFC price trend forecast, ... 128

Table 8.4.1: GA-Rule 2 scores for secondary series. 131

Table 8.5.1: ANTAS candidate Neural Network model for the LGFC 132

Table 8.5.2: Results for NN-2 on 1000 Out-of-Sample Forecasts 133

Figure 8.5.1: Cumulative Probability for a NN-2 correct forecast. 133

Figure 8.5.2: Histogram of probability of correct LGFC price trend forecast. 134

Figure 8.5.3: Probability changes for NN-2 out-of-sample compared to in-sample. 134

Figure 8.5.4: LGFC raw price movement over the NN-2 forecast horizon (37 days) 135

Figure 8.5.5: LGFC equity curve for the 1000 days experiments.. 136

Table 8.6.1: A control neural network configuration as a comparison to NN-2. 137

Figure 8.6.1: DRGA and Binary GA for GA-Rule 2 for the All Stock Index 137

Table 8.6.3: Multivariate Neural Networks for LGFC and LGFC Traded Volume. 138

Chapter 1

Learning Systems and Financial Modelling

This chapter presents a brief introduction to financial time series analysis, machine learning and neural networks. It provides an outline of the motivations and research goals, along with an overview of the research contributions and the thesis organisation.

1.1 Introduction

In recent years there has been a marked trend in the Artificial Intelligence (AI) community towards real-world problem solving. Techniques, inspired the broader ambition to produce more intelligent machines, are not only gaining acceptance in other fields of scientific research, but are also increasingly found in areas such as business, finance and industry. Moreover, there is a growing tendency within AI to develop, test, and refine such techniques within these domains. One of the best illustrations of this trend is the application of intelligent techniques in finance. Techniques such as Neural Networks, Genetic Algorithms and Fuzzy Logic have been used for financial forecasting, credit rating, customer profiling and portfolio management [Dav92], [Tayl93], [TrelGo93], [Debo94], [FeKin95].

Several factors have combined in order to accelerate this overall trend. From the business perspective, there is a genuine desire for automation at higher levels of company operations rather than simply automating a manufacturing processes. For example, the development of more effective management tools and decision support mechanisms are seen as one way in which a business can be made to be more competitive. Here AI methods are seen as a way of improving, and in some cases replacing, a domain expert's decision making, as well as offering a means for improving the efficiency and consistency of the decision process. This is a significant step beyond the type of automation that has been experienced in manufacturing, as has been pointed out for intelligent systems in finance [Ridl93]: "It is not about replacing clerks or secretaries but highly paid star traders".

Aside from the commercial impetus for developing intelligent solutions, the AI community has welcomed the challenge of building real-world systems. A traditional criticism of AI has been the brittleness of the solutions it produces, the suggestion being that AI systems have not scaled well beyond the relatively limited domains to which they have been applied. Moreover, the performance of an AI system is extremely sensitive to the representational choices made by its designer, and these are ultimately brittle in the face of inevitable deviations found in the real world. The argument is perhaps best summarised by [Drey79]: "the success of an AI system appears to be strongly correlated with the degree to which the problem domain can be treated as an abstract micro-world which is disconnected from the real-world at large". Only by designing systems that can cope with complex real-world situations can the AI community be said to have truly faced up to such criticism.

Against this background this thesis is concerned with the design of an automated system for financial time series forecasting. The objective is to explore the level of automation that can be achieved by a system for modelling a given financial time series with the minimum of human intervention. Two principal technologies form the basis of this investigation: feed-forward neural networks and genetic algorithms. Both techniques provide a general-purpose adaptive systems approach to problem solving [Wass89], [Dav93], [Tayl93] and have been used in a number of real-world financial applications [DeKin94], [FeTr94]. However, there are a number of technical issues that surround the application of these methods, particularly with regard to parameterisation. For example, the dominant issue in the application of feed-forward neural networks for time series forecasting is the question of how to achieve good generalisation. In Chapter 3 it will be shown that a network's ability to generalise out-of-sample is extremely sensitive to the choice of the network's architecture, pre-treatment of data, choice of activation functions, the number of training cycles, the size of training sets, the learning algorithm, and the validation procedure. Selection of determinants is another important factor in the development of a time series model, and one which is particularly hard when developing non-linear models. All of these aspects must be taken into consideration if the network is to perform well on data excluded from the training set. At present no formal framework exists for setting network parameters, and currently most researchers rely on extensive experimentation in order to isolate a good network model. Understanding these issues, and developing methods to tackle the problems they raise is the basis for the work presented in this thesis.

The remainder of this chapter is structured as follows: we firstly discuss the relevance of financial forecasting to AI learning systems in a broad sense. It is argued that applications such as financial forecasting present both unique challenges and opportunities for the AI community. It is shown that time series analysis (the basis of financial forecasting) and various formal definitions of machine learning have many aspects in common, and that the main distinction between the fields is the choice of vocabulary rather than any fundamental difference in their objectives. The principal learning paradigm used in this thesis is then summarised. Finally a detailed list of technical objectives is presented, along with a statement of the research background, research contribution and overview of the thesis organisation.

1.2 Adaptive Systems and Financial Modelling

Broadly speaking AI is concerned with creating computer programs that can perform actions comparable to human decision making. Traditionally an automated system is a machine with a fixed repertoire of actions, in which each of its actions corresponds to some well-defined task. Industrial automation, for example, addresses the process by which products are designed, developed and manufactured. The objective of an automated approach is to improve efficiency, increase quality, and reduce the time required to effect changes [Shap90]. In general, the most salient feature of such a system is that its behaviour is fixed, with each of its responses to outside conditions being pre-programmed by a system designer. Clearly, such a system pre-supposes the existence of an ideal response to a bounded set of outside conditions.

An adaptive system, on the other hand, attempts to limit the amount of *a priori* knowledge required in the system's specification. Instead the system *learns*, or adapts, its response to outside events according to its own experience. There are three key factors that make this approach particularly advantageous. Firstly, if the system can learn from past experience it is not necessary for the designer to specify all operating conditions under which the system is to perform. This is extremely useful when such conditions are unknown (for example in stock market trading). Secondly there is a flexibility associated with the systems developed, meaning that in changing business environments a system can adapt to new circumstances and adjust its response accordingly, without the need for costly reprogramming. Thirdly, there is the possibility of innovation. A learning system should always have the potential for discovery. A system free to make its own associations between the events that it experiences should be capable of finding relationships hitherto unknown. Indeed, it is this last possibility that provides the greatest advantage of an adaptive system over more conventional automated approaches, and it is this last factor that makes adaptive systems of particular relevance to financial time series modelling.

1.2.1 Financial Modelling: The Efficient Markets Hypothesis

The research presented in this thesis goes beyond a pragmatic demonstration of the potential of learning systems in finance. Firstly, any system (automated or otherwise) that attempts to forecast financial time series faces a significant technical challenge. Market trading, even in its most analytic forms, has been traditionally regarded as relying on intuitive and complex reasoning on the part of the human trader. The trader interprets and deciphers many factors surrounding the market of interest. The factors can be wide ranging and can vary over time. This kind of changing "structural" relationship has been interpreted as implying that the form of decision process required by market trading is not open to precise calculation, and therefore not open to mechanisation. Moreover, traditional economic theory has held that it is impossible to devise mechanised methods for predicting market movements at all [Fama70].

The so-called "Efficient Markets Hypothesis" (EMH) essentially states that the price of a financial asset fully reflects the available information related to that asset. In the case of the stock market, efficiency implies that stock prices equal the discounted value of expected future dividends. This is not to suggest that investors are making perfect forecasts of future dividends, but that they are making effective use of all information that is available [Shil87].

If capital markets are efficient in this sense, then changes in stock prices should be associated exclusively with new information leading to revisions in expected future dividends. This implies that information, once available, triggers a rapid process of adjustment, which re-prices the stock to its "correct" level, i.e., where it once more reflects all available information. This in turn implies that the movement of a price series is random in that it is based on future events. Unless a system can anticipate outbreaks of war (when Iraq invaded Kuwait oil prices rose), or political events (when Norman Lamont resigned as British Chancellor of the Exchequer share prices briefly rose on 25/5/93), or financial news ("Shares in Glaxo fell 13.5p after the company confirmed that it was facing allegations of "inequitable conduct" in a US court case" [Inde93]), it

is condemned to wait for information to arrive, and then react in a manner so as to incorporate this new information.

1.2.2 Learning Systems

The above outlines what is referred to as the *strong version* of the Efficient Market Hypothesis. The reason that the hypothesis is so relevant to *learning theory* is that it means no formal model for predicting market movements exists. The fact that no *a priori* model exists means that a "machine learning" solution is isolated from some of the usual charges made against AI. For example, we have mentioned that one common criticism of AI is the "brittleness" of the solutions it produces. Holland [Holl86] describes brittleness within expert systems as: "the systems are brittle in the sense that they respond appropriately only in narrow domains and require substantial human intervention to compensate for even slight shifts in the domain". Brittleness implies a narrow domain of application and a restricted set of outside events to which the system can successfully respond [GoKh95]. It includes the situation where a system designer can intervene in order to correct a system's output when outside events have changed. This ultimately relates to the Dreyfuss critique [Drey79] mentioned earlier, and amounts to the ability of the system designer to impose a solution on an abstract scaled version of the real-world problem. For automated financial time series analysis however, it is hard to justify any claim that the system's designer imposes a solution on a scaled version of the problem, in that no known solution exists. The Efficient Market Hypothesis suggests this. Moreover, if no known solution exists, and no theoretical models on which to base a solution exists, there are no grounds on which a human designer can intervene in order to *correct* the system's response. The autonomy of the system is made credible by the fact that no precise analytic methods exist on which to base the system's decision process.

Several issues are raised by the above. Firstly, the Efficient Market Hypothesis provides a significant challenge for a machine learning system. If a system is successful, then it must, to a large degree, take credit for having found the solution (i.e., since no *a priori* solution existed). However, if the Efficient Market Hypothesis is correct then no solution exists (automated or otherwise) and therefore the system is doomed to fail, no matter how sophisticated it is. In this sense, the choice of financial time series analysis (or any other field where existing theory states no solution exists) is high risk. In terms of this thesis, the risk associated with the Efficient Market Hypothesis is accepted and justified on the following grounds. Firstly, the Efficient Market Hypothesis provides a credible challenge to learning theory techniques precisely because of the risk of failure. Secondly, the approach that will be adopted will be sufficiently general-purpose that in the event of failure techniques developed within this thesis will have use outside that of financial time series modelling.

The technical content of this thesis is aimed at the design and investigation of automated adaptive systems. This investigation can be seen as independent from the overall goal of constructing an automated system for modelling financial price trends. That is to say, the experiments and the specific application of an automated system to a financial time series

problem makes up one section of the work reported and can be seen as an empirical test of the technical methods for automation that are developed within this thesis.

As a final comment, the possibility that the Efficient Market Hypothesis is wrong should also be discussed. If the hypothesis is wrong then there is no reason that a designer could not impose a solution on a given financial modelling task. As will be discussed in Chapter 2, there have been recent criticisms of the Efficient Market Hypothesis, and the possibility that it is untrue has been raised. However, this is a recent development, and at present no body of work yet exists with which to build a systematic approach to exploiting inefficiencies in markets, and therefore any success in modelling financial markets will provide a credible endorsement of the learning techniques used. Moreover, as mentioned above, for this thesis we concentrate on general-purpose techniques for the design of an adaptive system, and use financial modelling as the testing ground for the ideas developed within the main body of this work.

1.2.3 Technical Issues

This thesis concentrates on two main technologies – feed-forward neural networks and genetic algorithms. The goal is to design and implement an adaptive system for modelling an arbitrary financial time series problem. The system should *learn* to perform the task and have the facility to adapt and change its behaviour on the basis of historical data. At a technical level there are several immediate tasks that require attention. Firstly, the core of the modelling task will be attempted using feed-forward neural networks. This implies that careful consideration of the parameterisation issues surrounding the design of a neural network application will be required. Moreover, the design of the network topology, the activation functions used, the training methodology, the selection and pre-treatment of data will all have to automated (or fixed) according to some general design principles. This thesis will consider and develop methods for inferring network topologies, design experiments to test the inference mechanism, develop techniques for selecting and pre-treating data and construct methods for integrating data sets and time series models to form a single adaptive system for modelling a target financial time series. To assess the level of automation required will require a full understanding of the models offered by feed-forward neural networks. Chapter 3 conducts a systematic analysis of this style of neural network so as to gauge the relative importance of each of the available parameters with the specific aim of improving the likelihood of good generalisation.

Genetic algorithms will be examined as a means for tackling the problems raised in Chapter 3. In particular, methods for combining neural networks and genetic algorithms will be discussed as a means for automating the time series modelling process. However, as shall be described in Chapter 4, there are many issues surrounding the parameterisation of a genetic algorithm. In particular, the way in which mutation types, crossover styles, selection regimes, and the choice of representation interact with the fitness function to determine the likely success of the genetic algorithm search. It will be shown that the choice of representation for a genetic algorithm is fundamental to achieving good results, and that for real-world problems where little is known *a priori* about the fitness function, the choice of representation will always be problematic. To

address this issue we introduce a new form of genetic algorithm that includes randomly selected multiple representations in the course of a run. By adopting a search strategy that includes multiple representations it is possible to probe the space of search algorithms with respect to the search problem. Such algorithms are shown to out-perform the binary Holland genetic algorithm on a range of standard test functions. Despite this innovation, it will be argued that the direct application of a genetic algorithm to neural network parameterisation potentially raises more problems than it solves. In particular, in the absence of a precise formulation of *generalisation*, the combined complexity of a genetic algorithm and neural network runs a significant risk of over-fitting a given data set, and as such a less direct use of genetic algorithms to network parameterisation may be of greater benefit. In Chapter 5 we introduce a new method for investigating neural network parameterisation based on genetic algorithms.

Chapter 5 introduces a technique that uses genetic algorithms as a means for artificially generating test learning problems, in the form of a generated time series, for feed-forward neural networks. The test learning problems have known network solutions, in that there is a known approximately Minimally Descriptive Network architecture, with known weights, that can map the generated learning problem. The technique, *Artificial Network Generation* (ANG), allows three subsequent lines of inquiry; firstly it becomes possible to assess the sensitivity of a neural network's generalisation ability with respect to a given network's architecture in a precise manner. Secondly, it is possible to assess the relevance of Occam's Razor as a heuristic for network architecture design for promoting good generalisation. And thirdly, it becomes possible to validate in an objective manner automated methods for designing a network's architecture for an arbitrary learning problem. Specifically, ANG is used to generate a suite of test learning problems. These are used to conduct experiments in which networks using different architectures are trained on the ANG learning problems. In each case the network using the approximately minimally descriptive architecture out-performs all other networks in terms of generalisation (measured via an iterated out-of-sample forecast). From this it is possible to provide clear statistical evidence in favour of Occam's Razor as a design principle for a network's architecture.

Having established Minimally Descriptive Network architectures as providing a means for promoting good generalisation we focus our attention towards developing a method for inferring minimally descriptive architectures for a given learning problem. Specifically, we introduce a new method for network pruning. The technique, *Network Regression Pruning* (NRP), is a network pruning method that attempts to derive an approximately Minimally Descriptive Network architecture by starting from what is considered an overly large network. NRP differs radically from conventional pruning methods in that it attempts to hold a trained network's mapping fixed as the pruning procedure is carried out. In this sense the technique attempts to infer redundancy in the network's architecture by isolating the exploration of architecture space from that of weight space (in contrast to existing pruning methods which attempt to explore both simultaneously). As pruning proceeds a network *regression pruning error profile* is generated. The profile records the Mean Square in-sample Error of each network produced via each stage of the pruning procedure. It is shown that catastrophic failure of the network, marked by an

exponential increase in the in-sample Mean Square Error, approximately coincides with the Minimally Descriptive Network for the given problem. This hypothesis is tested using ANG. On the basis of these results an automated network complexity[1] inference procedure is derived. This technique is shown to be extremely successful at isolating network complexities on a range of test problems generated using ANG.

The final phase of the work described within this thesis consists of a detailed investigation of all the techniques developed in terms of their performance on a real-world financial time series modelling problem. In particular we shall describe an extended series of experiments in which an Automated Neural net Time Series Analysis system (ANTAS) will be used to model a gilt futures price series, the Long Gilt Futures Contract. The results of all experiments, both in- and out-of-sample will be described in some detail, with the focus of attention on the way in which the automated design procedures developed within this thesis respond under real-world conditions. The next section describes the broader context of this thesis, in particular the association between time series modelling and machine learning.

1.3 Time Series Analysis

There are a number of reasons that make time series analysis and machine learning particularly compatible. The surprising fact is that the essential objectives of both fields are extremely similar. In both fields the major concern is developing a system that can generalise future events by taking samples of past events. This is true for both time series analysis and most definitions of "machine learning". In this section we explore the link between the two paradigms, and suggest that both fields offer approaches that are mutually beneficial in terms of the techniques and methods they employ.

1.3.1 Fundamentals of Time Series Forecasting and Learning

Analytic approaches to time series forecasting rely almost exclusively on the ability of the forecaster to identify some underlying relationship within the past values of a target time dependent process in order to make explicit some formulation of that relationship. The formulation is intended to capture the underlying mechanism that drives the target process. If this can be achieved it then becomes possible to either explain the process, or to predict its future values. Take the following list of criteria identified by Chatfield [Chat89] as possible objectives for time series analysis:

- **Description:** the need for some high-level description of the main properties of the target series, for example, dominant features such seasonality or annual trends.
- **Explanation:** finding connections between two or more variables, where the movement of one affects the movement of the other. This might also refer to different time periods within the same series.

---

[1] A network's complexity is taken to be its number of weights [Chapter 3].

- **Prediction:** the wish to find future values.

These objectives outline a representative set of criteria involved when attempting to formulate a time series relationship using traditional analytic techniques. Now compare the above with the criteria as set out by Michalski et al. [MiCM86] for assessing the performance of a learning algorithm. Michalski would be considered as a mainstream AI theorist:

- **Validity**: how well the representation fits the real case.
- **Abstraction Level**: the scope, detail and precision of the concepts used in the representation, its explanatory power.
- **Effectiveness**: the performance of the representation, how well it achieves its goals.

This list of validation criteria for machine learning applies precisely to Chatfield's list of time series objectives. For "Validity" in Michalski's list read "Description" in Chatfield's. Both require representations (or descriptions) of the target concept (or series). In the time series case it is tacitly understood that the representation should attempt to match the real case as accurately as possible. For "Abstraction Level" we have "Explanation", using machine learning terminology, the level and scope of detail identified in the "Abstraction" phase of time series analysis will determine the accuracy with which the series is ultimately explained. Lastly, for "Effectiveness" read "Prediction". In forecasting "Effectiveness" is readily interpreted as a measure of the reliability and accuracy of the forecasts made. In machine learning "Prediction" would correspond to the generalisation ability of the machine's constructed representation. Machine learning as defined above covers a far wider spectrum of event than that of time series analysis, but forecasting can certainly be seen as included in the definition.

A more direct comparison between learning and forecasting is made possible if we take an older definition of machine learning. Inductive inference as established by Gold [AnSm83] denotes the process of hypothesising a general rule from examples. Angluin and Smith [AnSm83], give the example of trying to guess the next number in a numerical sequence 3,5,7, and the possible alternatives that may be offered (next number 9 and the rule being odd numbers starting from 3, or the next number 11 with the rule odd prime numbers). This example has a clear resonance with time series analysis.

Strictly speaking there are differences between induction and traditional AI learning. Angluin and Smith [AnSm83] cite Webster's dictionary to make the point. "To learn" is "to gain, knowledge, understanding or skill by study, instruction, or experience". In contrast "induction" is "the act, process or result of an instance of reasoning from a part to a whole, from particulars to generals, or from the individual to the universal". They add that AI learning has been more concerned with cognitive modelling than actual inference. A survey article by Dietterich et al [DLCD82] divides learning into four areas: rote learning. learning by being told, learning from examples, and learning by analogy. Of these, learning from examples has the greatest overlap with induction and has the most relevance to this thesis. It can also be said to have the largest crossover with time series analysis. Perhaps the most striking testimony to this is the fact that some techniques have been shared directly by both communities. One of the earliest examples of

shared techniques is the use of Bayesian inferencing. Chatfield [Chat89] describes Bayesian forecasting (developed by Harrison and Stevens [HaSt76]), which uses a combination of Bayesian analysis and linear modelling to form an adaptive time series modelling technique. Bayesian analysis is used to update parameters within the model as more observation become available. Bayesian *learning* on the other hand has been described by Duda and Hart [DuHa73] in which Bayesian analysis is used to continually update parameters within a classification system.

A more recent example of the crossover between time series analysis and learning systems is the use of feed-forward neural networks. Feed-forward neural networks learn from examples and, as will be described in Chapter 2, have made a significant impact on the time series modelling community. In particular, feed-forward neural networks are seen as offering a general-purpose paradigm for non-parameteric non-linear regression. Moreover, techniques developed as part of statistical time series analysis have been adopted by the neural network community as a means for validating trained neural network models. Ultimately, it is the similarity between time series analysis and machine learning that makes time series analysis an extremely convenient domain in which to test and develop machine learning techniques. In Chapter 2 we review the progress of neural nets and financial modelling and make the case for selecting feed-forward neural networks as the basis for developing an automated financial time series analysis system. Many of the techniques developed within this thesis are aimed at inferring, testing and validating neural network models. As such it is important to view this work in the context of the development of the neural network paradigm. In the next section we provide a brief introduction to the area so as to set the background to much of the work included in later chapters.

1.4 A Brief History of Neural Networks

The mathematical modelling of neurons started in the 1940s with the work of McCulloch and Pitts [McPi43]. Their research was based on the study of networks of simple computing devices as a means of modelling neurological activity in the brain. During the same period Hebb researched adaptivity and learning within these systems [Hebb49]. Most of the early research was biologically inspired, and did not directly relate to devising new methods of computation. The focus on computational ability came later in the 1960s with the studies conducted by Rossenblatt [Rose62]. His work concentrated on the study of *perceptrons*, or as they might be called today *single-layer feed-forward networks*. Central to all of this research, and subsequent work, is the idea of a lattice of artificial neurons, or nodes, or more recently units.

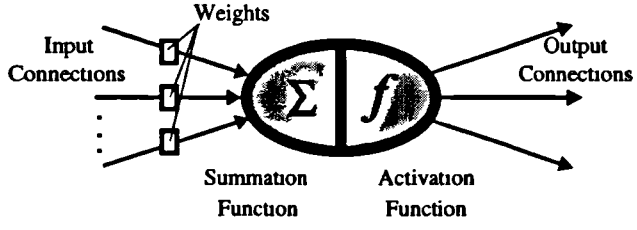


Figure 1.4.1: A Conventional Node.

Typically, the artificial neuron is an analogue of the biological neuron, which models features of the biological counterpart without attempting to faithfully represent all of the internal

mechanisms of a cell. Figure 1.4.1 depicts an example of a node. The basic features consist of input-output connections, a summation device for input activity, and an activation function applied to the summed input activity so as to determine the node's own level of response, and hence its output.

As in the brain, artificial neurons can only perform useful functions by acting in concert. Many connections, each of which carries an associated weight, are established between nodes which are typically formed in layers. Figure 1.4.2 depicts an example neural network (Note: although almost always depicted as a separate layer, the input layer performs no transformation on its incoming signal and serves just to 'fan-out' the input vector to other units). The weights act as scalar multipliers to the activation pulse that passes along a connection between nodes. Typically, a multi-layer feed-forward net processes information by the forward pass of an activation pulse (node-to-node, layer-to-layer) in response to a stimulus at the input layer. The network's response is then the activation vector that appears at the output layer. In this respect the neural network can be seen as a functional di-graph mapping an input space to an output space, generally of the form $R^n \rightarrow R^m$ for real-valued vectors, with n the number of input nodes and m the number of output nodes. In this case, each network topology and fixed set of activation functions, can be seen as bounding a large class of functions which map from $R^n \rightarrow R^m$. In order to realise a specific map some scheme is required for setting the weights. For neural networks this introduces the notion of a training or *learning* algorithm, so called because in most instances the weights are set by specifying a *training* set of paired input-output instances that the network must learn to map. In this version, so-called *supervised* learning, the learning algorithm explores the set of possible weight values in order to eliminate configurations incompatible with the examples of the desired map. In this sense the "knowledge" of the trained neural network can be said to reside in the weights.

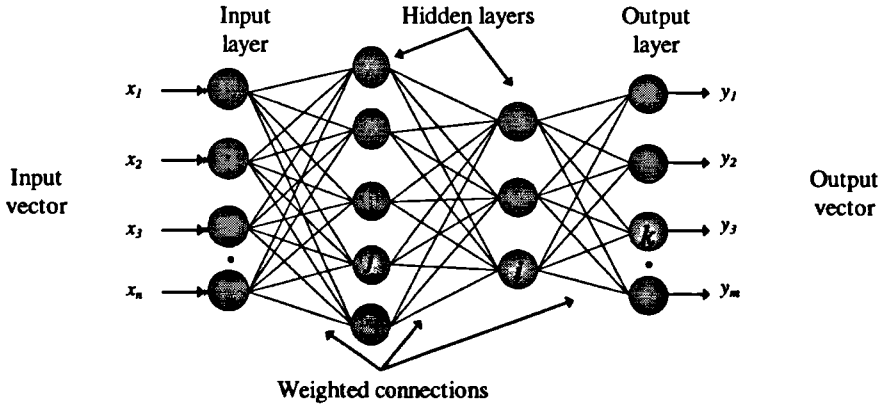


Figure 1.4.2: Four-Layer Neural Network.

1.4.1 The Development of Neural Net Techniques

The perceptron is a learning algorithm for single layer neural networks using linear threshold activation functions. Rosenblatt [Rose62] showed how a simple learning rule, the Perceptron Rule [Hebb49] could be used to train a perceptron to perform pattern recognition tasks. The process of training consists of presenting a fixed set of input-output pairs to the network and

iteratively adjusting the weights in order to improve performance by reducing a fixed cost function (a measure of the output error). It was shown that the perceptron learning algorithm could learn any linear separation from examples [Pape61], [Rose62]. In the 1960s, work in this area focused on the study of threshold logic and the computational abilities of assemblies of such devices. It is the popular view that interest declined when the limitations of single layer networks were made explicit [MiPa69]. The classic example of this is the inability of a perceptron to learn the class of linearly inseparable functions. Such functions when graphically represented cannot be dichotomised with a single hyperplane. Accordingly, a perceptron using a single linear threshold activation function is unable to dissect the positive and negative instances of the function. It was pointed out at the time [MiPa69] that the limitations could be reconciled if the network made use of more layers, but that an effective training algorithm would be problematic[2].

In the 1970s research concentrated on the more biological aspects of neural networks, in particular the use of the paradigm as a neurological metaphor. It concentrated on finding biologically credible models of distributed processing and self-organisation [Gros82]. Work by Kohonen and others on feature extraction [Koho89] and clustering techniques was also prominent during this period. These models differ from the supervised learning presented above in that the networks are trained on a set of inputs, without the corresponding output pairs. The algorithm assumes that there exist patterns within data which carry meaning (such as clusters) but for which no explicit response is available. The *self organising map* learns to cluster input vectors by adaptively changing its connection weights following a simple set of rules. The learning process results in a set of associations between subsets of input vectors and specific output nodes. This is known as *unsupervised* learning.

In the mid 1980's the resurgence of neural network research activity is generally attributed to several independent events [Sont93]. One factor was the work by Hopfield [Hopf82] on the design of associative memories, and later the solution of optimisation problems, using a special class of recurrent single-layer networks. Other factors include the introduction of the Boltzmann machine [AcHS85] and, more prominently, the backpropagation learning algorithm [RuHW86], [Park84], [LeCu85], [Werb74].

Backpropagation, a supervised learning scheme (see Chapter 3 for a full derivation), is the most well known and widely used form of neural network training. It is essentially a variation on the perceptron and marks a return to the feed-forward nets of Rossenblatt. Two main differences exist: the network consists of a fixed architecture but this now includes hidden layers, and the activation function is made continuous. Training consists of a gradient decent procedure through the weight space in order to minimise a pre-set cost function. It has subsequently been shown that if the network makes use of a bounded, continuous, non-decreasing activation function, a three-layer network (one hidden layer) is capable of learning an arbitrary continuous mapping between input and output space, provided no restriction is placed on the size of the hidden layer [Cybe89],

---

[2] The so called loading problem for threshold neural networks has subsequently been shown to be NP-complete [Judd90]

[HoSW89], [Horn91], Moreover, for finite training sets of N input-output pairs it is quite simple to show a network using N-1 hidden nodes can recreate the necessary look-up table [SaAn91]. In Chapter 3 we demonstrate this ability by deriving an explicit formula for setting the weights of a network to so as to map a finite data set to arbitrary precision.

1.4.2 More Recent Issues

Since the mid-eighties neural network research has undergone a rapid period of expansion [Tayl93]. One estimate puts the number of actual neural network models at over 100 [Trel89]. Both supervised and non-supervised neural networks are finding increasing applications in a variety of domains [Tayl93]. The class of learning procedures covered by backpropagation has proven to be remarkably powerful and robust. It has been shown to be capable of state-of-the-art performance in signal processing and other statistical decision tasks [FCKL90]. The proximity of learning from examples, or supervised learning, to time series analysis has also meant that neural networks have found increasing usage as forecasting tools. In particular, networks have been used for predicting trends in stock prices, market analysis, exchange rate forecasting, as well as for credit and insurance risk assessment and fraud detection, [Scho90], [WeRH91], [King93]. From the author's personal experience it is known that most of the major UK high street and merchant banks are involved in some form of applied neural network research.

One of the more active research themes has been neural network parameterisation. This has involved the introduction of new training algorithms (network growing and network pruning) as well as more rigorous methods for assessing a neural network performance, particularly in terms of generalisation. On a related issue attention has focused on the question of *learnablity*. For a function to be learnable by a given net a set of weights must exist that realises the function. From this it is natural to ask questions as to the relationship between the size of a training set and the number of free parameters available during training? We shall return to these issues in Chapter 3 where we make explicit the number and variety of parameters involved in the design of a neural network application, and assess the relative sensitivity of a neural network's performance to specific parameter choices.

1.5 Thesis Overview

The remaining sections of this chapter provide a brief summary of the research objectives, research contributions as well as the overall structure of this thesis.

1.5.1 Research Objectives

The research goal of this thesis is to investigate and develop techniques for automating the application of feed-forward neural networks to time series analysis problems. The techniques are to be validated by constructing an automated time series analysis system applied to a financial time series forecasting problem. The techniques introduced aim to provide a general-purpose applied methodology for developing neural network applications for arbitrary learning problems. The overall aims can be summarised as:

i) Investigate the relationship between neural network parameters and their effect on over-fitting and generalisation.

ii) Develop automated methods for parameterising neural network models for times series analysis so as to promote good generalisation.

iii) Investigate genetic algorithm techniques as a means for addressing some of the issues raised in ii).

iv) Develop automated methods for the selection and pre-processing of data for a neural network time series application with an emphasis towards financial time series modelling.

v) Test and validate techniques that are developed.

vi) Implement an automated adaptive system using neural networks for time series analysis.

vii) Provide an empirical investigation of the system developed on a financial time series, along with a full analysis of the system's performance and relevant conclusions on the methods developed.

1.5.2 Thesis Structure

In order to meet the above list of objectives this thesis is structured as follows:

Chapter 2 investigates the application of adaptive systems in financial modelling with specific reference to the Efficient Market Hypothesis. This chapter makes the case for using feed-forward neural networks as the core technique for an automated adaptive system for financial time series analysis.

Chapter 3 investigates the problems associated with the parameterisation of feed-forward neural networks so as to ensure good generalisation and to prevent over-training. This chapter conducts a detailed examination of these issues so as to highlight and assess the problems they raise, and concludes by suggesting the use of genetic algorithms as one method for probing these issues.

Chapter 4 investigates genetic algorithm as an optimisation technique. This chapter provides a careful examination of genetic algorithm parameterisation with respect to an arbitrary search problem. The investigation is aimed at providing general genetic algorithm design principles and concludes with a recommendation as to how best to use genetic algorithm to address the neural network parameterisation problem.

Chapter 5 introduces the Automated Network Generation (ANG) procedure. This uses genetic algorithms as a means for generating test learning problems for feed-forward neural networks. A careful examination of Occam's Razor as a design principle for network architecture is conducted. A method, Network Regression Pruning (NRP) is introduced as method for inferring network complexity for a given learning problem. The technique is validated using ANG.

Chapter 6 combines the methods developed in chapters 3, 4, and 5 in order to implement the Automated Neural network Time series Analysis System (ANTAS). The chapter provide the

implementation details of ANTAS, describing each module within the system and the control process for automated time series modelling.

Chapter 7 describes the target financial data series that is to be used to test ANTAS. This chapter describes the Long Gilt Futures Contract (LGFC) and other trading series that will be used by ANTAS (in Chapter 8) in order to formulate a predictive model of the LGFC. The chapter provides a description of this data series in the context of financial forecasting, and in terms of the Efficient Market Hypothesis. Details of specific modules within ANTAS for handling aspects of this data series are also given. This includes automated methods for setting forecast horizons, and dealing with Moving Averages of the target series.

Chapter 8 presents gives a step-by-step analysis of ANTAS as applied to LGFC forecasting. This includes the specifics of the modelling process within ANTAS, and an analysis of the decision process behind the model construction phase. A full analysis is given of an extended series of out-of-sample experiments using ANTAS to model the LGFC. ANTAS is applied in a single run to 1000 out-of-sample experiments. It is shown that the system isolates a deterministic signal within the LGFC, to a confidence level of over 99%, yielding a prediction accuracy of over 60%. These results are analysed in terms of the ANTAS design methodology, and their general significance in terms of the Efficient Market Hypothesis. Chapter 9 concludes the thesis with a general discussion of the results achieved, providing both criticisms and suggestions for future work.

1.5.3 Research Contribution

The main contribution of this thesis to the development and understanding of adaptive systems are stated below.

In the following descriptions technical terms are used which have thus far not been introduced. The reader should refer to the relevant chapter for explanations and definitions.

1. A rigorous analysis of the modelling behaviour of feed-forward neural networks. A formula is derived for setting a feed-forward network's weights in order to map an arbitrary finite set of input-output pairs to arbitrary precision. This provides a simple demonstration that feed-forward neural networks using sigmoid activation functions are universal functional approximators for finite sets. A detailed analysis in which the effects of various network training parameters on the level of generalisation achievable by a trained neural network is given. A comprehensive list of network parameterisation problems is provided so as to make explicit the decision process required in order to automate the application of a feed-forward neural network for a given learning task [Chapter 3].

2. A detailed analysis of the genetic algorithm search technique. An investigation into the relationship between genetic algorithm parameterisation and likely search success is provided. It is shown that it is the combination of representation and traversal operators that define an algorithm's *view* of a given search problem, which *gives rise* to a fitness landscape. In this sense a fitness landscape only exists in terms of a given search algorithm, and that notions of landscape

difficulty are therefore algorithm dependent. It is shown that genetic algorithms have attractive features in terms of the way a landscape can be adjusted, and that this can provide a simple means for applying multiple search strategies to a given search problem. A new style of genetic algorithm is introduced that takes advantage of this observation. The Multi-Representation genetic algorithm uses multiple randomly selected representations during the course of a run, making use of a *transmigration* and *transmutation* operators. Multiple representations are achieved through base changes in the genotype of the individuals within a population. It is shown that contrary to conventional genetic algorithm analysis there is no formal justification for preferring binary representations to those of higher base alphabets. A series of experiments demonstrates and compares Multi-Representation GAs to the conventional binary Holland genetic algorithm on a range of standard test functions. It is shown that the Multi-Representation genetic algorithms out-perform the conventional model on this set of test problems, and compare favourably to other styles of genetic algorithm reported in the literature.

3. A new approach for analysing neural networks is introduced. The approach is based on a method for artificially generating test learning problems that have know neural network solutions (i.e., a known approximately Minimally Descriptive Network architecture and weights that are capable of mapping the test problem). The technique, Artificial Network Generation (ANG), can be used to generate a suite of test learning problems which can then form the basis of controlled experiments into neural network parameterisation.

4. Using ANG a series of experiments is conducted in order to test Occam's Razor as a neural network architecture design principle. Statistical evidence is provided that supports the use of this heuristic. Specifically it is shown that Minimally Descriptive Networks out-perform larger networks in terms of generalisation on the range of test problems generated using ANG.

5. A new method for neural network pruning is introduced. Network Regression Pruning (NRP) is a network pruning method that attempts to derive an optimal network architecture by starting from what is considered an overly large network. NRP differs radically from conventional pruning methods in that it attempts to hold a trained network's mapping fixed as the pruning procedure is carried out. As pruning proceeds a network regression pruning error profile is generated. The profile records the Mean Square in-sample Error of each network produced via each stage of the pruning procedure. It is shown that catastrophic failure of the network, marked by an exponential increase in the in-sample Mean Square Error, approximately coincides with the Minimally Descriptive Network for the given problem. This hypothesis is derived using ANG. On the basis of these results an automated network architecture inference procedure is inferred.

6. Using NRP a novel method for applying genetic algorithms in order to automate the application of a neural network is introduced.

7. The design of an automated adaptive system for time series analysis is introduced. The system, Automated Neural network Time series Analysis System (ANTAS), combines neural networks and genetic algorithms in order to automate a time series modelling task.

8. Methods are introduced for comparing time series models over a range of input conditions. The technique is employed by ANTAS as a means for selecting neural network models for a given learning problem. The technique is demonstrated in the context of financial time series modelling and specifically for forecasting price trends in the Long Gilt Futures Contract.

1.6 Summary

This chapter has provided a general overview of the work contained in this thesis. A detailed list of research goals and achievements have been described. A brief introduction to the principal research area has been given. This includes a brief outline of the connection between machine learning and time series analysis, as well a brief overview of the principal machine learning technique that will be investigated through-out this thesis, namely neural networks. The next chapter takes a more detailed look at the problems associated with financial time series modelling, and reviews and assesses existing research into the area of neural networks and financial times series modelling.

Chapter 2

Adaptive Systems and Financial Modelling

This chapter reviews the application of adaptive systems in financial modelling. Specific reference is made to the Efficient Market Hypothesis and the technical problems its raises in terms of financial time series analysis. A survey of financial Neural Network applications is provided, along with a summary of empirical assessments of Neural Networks for time series modelling. Examples of Genetic Algorithms applied in finance are also included.

2.1 Financial Modelling

Economic time series are often cited as examples of randomness [FaSi88]. Since there are no formal criteria for deciding whether a series is truly random, this really amounts to an empirical statement regarding the lack of success in predicting such series. It is questionable whether something like the Efficient Market Hypothesis is a product of this lack of success, or that its existence has prevented more research into modelling such series. Either way, there are several reasons why traditional modelling techniques have had difficulty in coming to terms with economic and financial time series analysis. Firstly, there is no rich body of theoretical or empirical research extending over the centuries on which to base a modelling approach [Dick74]. Secondly, the laboratory for testing and applying any theoretical advance is the real world, with all the difficulties associated with real-world experimentation. Only very recently have computers provided sufficient power to enable some means of running realistic experiments. A third reason relates to the underlying nature of what is being modelled. Share prices, for example, are influenced by a range of factors, some known, some unknown, some quantifiable, some objective and some subjective. They can range through interest rates, exchange rates, company profits, political events, economic forecasts, consumer preferences, strikes, rumours and human frailty [Dick74]. The list could be endless. How to isolate significant determinants is a major factor in any forecasting system. In the next few sections we highlight some of these issues in terms of the technical problems associated with financial modelling. We start with some of the problems associated with traditional statistical approaches to financial/economic modelling.

2.2 The Problems with Financial Modelling

Traditionally the basic methodology for model building for both financial and economic time series has been statistical [OTW91]. Clearly this encompasses a large range of possible methods and activities, however certain general features can be summarised. Model building is generally carried out by a domain expert [Tong90]. On the whole, statistical model building is not a process that can be easily automated it can require expert interpretation and development at every stage. All considerations increase the time needed to construct, evaluate and test a model, and consequently limit the number of determinants that will be considered. Further to this is the

sensitivity of the techniques themselves. For the most part statistical model building requires certain regularities in the target data. For example, the series must be stationary[1] [Chat88], or the process should have significant correlation coefficients between specific lags. Such factors serve to set conditions of use, or criteria that must be satisfied in order to apply the techniques. If the series fails to satisfy any of these requirements then this can prevent the model from ever being constructed. These technical requirements have hindered traditional model building in both economics and finance. Moreover, *most* statistical techniques are linear [Tong90] and yet most economic and financial relationships are hypothesised as being non-linear [OWT91]. Non-linear statistical analysis is, if anything, more demanding in terms of the criteria that must be satisfied to apply the techniques, and again requires expert interpretation of the various parameter choices [Tong90].

The dominance of linear statistical methods is best illustrated by the quantitative analysis techniques that have been developed over the last 30 years. Modern Portfolio Theory (MPT), for instance, was developed by Markowitz [Mark59] as a means of stock selection for an investment portfolio. Markowitz emphasised that investors should maximise the expected return on their investments for a fixed level of risk, the underlying tenet being that pricing of assets is inherently related to risk. MPT proposes that the variance of returns of a portfolio should be used as a measure of its risk, and the covariance of a stock's return with respect to the portfolio as a whole should be a measure of how diversifying that stock is. This formulation led to the solution of portfolio selection in terms of a quadratic optimisation problem, yielding one of the first systematic approaches to the investment selection problem. This approach is very much at the core of a large fraction of portfolio management systems today. The Capital Asset Pricing Model (CAPM) [Shar64], [Lint65], [Moss66] is one of a number of models that grew out of MPT. It further quantifies the relationship between risk and related return of an asset by modelling the return of an asset as a linear function of the return of the market as a whole. The strength of this relationship, the so-called beta value, is now a standard statistic reported for assets. The assumption of linearity has been a central theme in the resent criticisms of these methods and has been forwarded as an explanation for their lack of empirical success [Malk90].

2.2.1 Fuzzy Rationality and Uncertainty

Another technical difficulty associated with financial/economic modelling is that many of the rules which (appear to) successfully describe the underlying processes involved are qualitative, or fuzzy, requiring judgement, and hence by definition are not susceptible to purely quantitative analysis [OTW91]. For example, the techniques employed by "chartists" [Malk90] are inherently fuzzy, relying on an expert to decide when certain conditions are met in the shape of price movement. This, to some extent, has prevented rigorous academic scrutiny of their methods.

---

[1]Broadly speaking a series is stationary if there is no systematic change in the mean (no trend), if there is no systematic change in the variance, and if strictly periodic variations have been removed.

A far worse reason than the technical difficulties sketched above is the theoretical resistance to both economic and financial modelling. The foundation of the theoretical scepticism is based on information exchange, or, more precisely, the way information is generated and consumed, and relates to the idea that structural relationships between determinants and the target process can change over time. For instance, it is hypothesised that the changes can be inherently unpredictable, possibly abrupt, and even contradictory in nature [OTW91]. The classic example of this is the fact that one month a rise in interest rates can strengthen sterling, whilst the next month a rise can weaken it. The phenomenon of unstable structural parameters makes the development of fixed models almost impossible. Any relationship that can be established is always at the mercy of shifts in future economic and financial policy.

Economic uncertainty, as opposed to risk, was suggested by Knight [Knig21], [Dav91] to be central to economic activity. Uncertainty pertains to future events such as financial crisis and shifts of policy regime not susceptible to being reduced to objective probabilities. Risk, on the other hand, is something that is estimated and frequently used within all forms of economics. Despite its ubiquity there is no universally accepted definition of risk, but in most instances it is taken to refer to something like the standard deviation of the target series [Dick74]. This gives a measure of volatility, and accordingly a measure of the possible consequences of an investment. This, however, is not a prediction. It has been argued [Melt82], [Dav91] that events not susceptible to probability analysis are excluded from rational expectation models of decision making and therefore from optimal diversification of risk.

Rational expectations has appeared in economics literature since the early 1960s, but has only been a major theme since the mid-1970's [PeCr85]. The theory relates to the way some economists regard the present as affecting the future. It is based on the micro-economic idea that people are rational, and therefore attempt to do the best they can. They formulate their views of the future by taking into account all available information, including their understanding of how the economy works. Thus it relates to the way economists have modelled the future on the basis of what people expect from that future. If uncertainty, of what everkind, cannot even be reduced to a probability then it cannot be rationally accounted for, and therefore play a part in a risk-strategy for the future. Accordingly, proponents of this view have then pointed out that most economic/financial forecasting has not provided a basis for reliable predictions; nor indeed have they provided convincing explanations for them, [Dav91] and the reasons for this are the inability to deal with uncertainty, and the fact that they are at present restricted to the narrower consideration of risk. Note, that this line of reasoning ultimately concludes that economic and financial systems are unforecastable and essentially random.

The idea of uncertainty and inherent limitations of standard techniques is representative of a general mood in present day econometric forecasting [OTW91]. The so-called Lucas critique [Lucu76] has been applied to most forms of macro-econometric modelling. Advanced in 1976, this hypothesis claims that parameter values are not robust with respect to changes in policy regime. Ultimately, the validity or otherwise of this critique is again empirical. Theoretically it is certainly true – new policies cannot be predicted, and therefore what consequences they may

imply must be random. Asset price data, for example, is considered a special case of the Lucas critique, and traditionally has been seen as a powerful example of it. Most work relating to statistical analysis of asset price movement has concentrated on demonstrating the principle at work, and hence supporting the view that no mechanised profitable trading strategies exist, and that the series themselves are random. Uncertainty in asset prices relates to the Efficient Market Hypothesis. Efficiency, if true, is certainly another reason for the poor performance of traditional modelling techniques. This theory is naturally of fundamental importance to any technique that is attempting to refute it, and it is therefore worth discussing the current status of the theory from the economics perspective.

2.2.2 Efficient Markets and Price Movement

The earliest descriptions of market dynamics related mostly to psychological factors rather than any analytic attempt to explain the movement. Justifications for movement were given in terms such as: investors overacted to earnings, or where unaffected by dividends; or, there were waves of social optimism or pessimism; or buying was subject to fashion or fads [Shil87]. This has changed over the last two decades. Statistical evidence amassed over this period has been widely interpreted as implying that markets are efficient [Falm70], [Malk85]. This amounts to the statement that the return on an asset can not be forecasted, and that all information about the asset is efficiently incorporated in its current price. Moreover, no one can know if the asset is a better or worse investment today than at any other time [Shil87].

This Efficient Market Hypothesis has been refined into three sub-hypotheses: the strong, the semi-strong and the weak. This is summarised in Table 2.2.1.

Efficient Market Hypothesis types	Impossible to forecast a financial instrument based on the following information.
Strong	i) Insider information (publicly unavailable), ii) Publicly available information (e.g., debt levels, dividend ratios etc.), iii) Past price of the security.
Semi-strong	ii) Publicly available information (e.g., debt levels, dividend ratios etc.), iii) Past price of the security.
Weak	iii) Past price of the security.

Table 2.2.1: Versions of the Efficient Market Hypothesis

2.3 Evidence Against the Efficiency Hypothesis

All of the above severely questions the likely success of any attempt, automated or otherwise, to model financial/economics time series. However, recent questions have been raised about aspects of the hypothesis. Evidence against the strong version of the theory was first put forward almost twenty years ago [Jaff74]. Jaffe's studies revealed that insiders having knowledge regarding the state of companies (e.g. knowledge that the company is planning a take-over) could use inside information to manipulate the market to their own ends. The validity of this form of market inefficiency is now widely acknowledged, and has led to preventative legislation by market regulatory bodies. In many countries insider trading has now become a serious criminal offence. A recent study by Meulbroek [Meul92] goes so far as to point out trading patterns

associated with insider dealers, and shows that it is in fact possible to detect such trading from patterns within the series alone. This would appear to strongly reject the idea of total randomness, and even suggests a means by which a third party could profitably trade by following single price series movements.

Throughout the 1980s, many studies have also presented evidence against the semi-strong version of the hypothesis. These studies discuss instances where publicly available information related to securities can be used to forecast market movements with a much higher level of predictability than a random walk model would imply. An example of this is so-called *Calendar Effects* [Fren90], where securities rise and fall predictably during particular months and on particular days of the week. Monday returns are on average lower than returns on other days [Hirs87], [Cast91]. Returns are on average higher the day before a holiday, and the last day of the month [Arie87]. In January, stock returns, especially returns on small stocks are on average higher than in other months [Cast91]. Much of the higher January returns on small stocks comes on the last trading day in December and in the first 5 trading days in January. Further evidence against the semi-strong version is provided by a range of studies which examine the effect of dividend yields, earnings/price ratios and other publicly available information on the predictability of stock returns. Basu [Basu77] found that earnings/price ratios are also very good predictors of future prices. This type of analysis of securities is termed *fundamental analysis* in the investment community and has been making an increasing impact over the last four years [Ridl93].

More recently there has been evidence against the weak form of the Efficient Market Hypothesis in which past prices alone have been used to make successful predictions of future movements of securities. For example, Lo and MacKinlay [LoMa90] reported positive serial correlation in weekly returns of stock indices where a positive return in one week is more likely to be followed by a positive return in the next week. Jegadeesh [Jega90] has found negative serial correlation for lags up to two months and positive serial correlation for longer lags for individual securities.

Some of the most convincing evidence against the weak form of the Efficient Market Hypothesis is from research studying the predictability of markets using technical trading rules [BLLB91]. These trading rules have been used for many years by traders practising technical analysis, but have only recently been subjected to academic examination. Brock et al [BLLB91] have tested some of the most popular trading rules, in particular the sorts of rules employed by chartists. They applied the techniques to 90 years worth of data from the Dow Jones index: 20 versions of the *moving average rule* (when a short-term moving average rises above a long-term moving average, buy) and six versions of a *trading-range break rule* (buy when the index goes above its last peak, and sell when it goes below its last trough). Contrary to previous tests they found that both types of rules work quite well. Buy signals were followed by an average 12% return at an annual rate and sell signals were followed by a 7% loss at an annual rate. This type of result contradicts all forms of the Efficient Market Hypothesis, but how far it goes in developing a deeper understanding of market dynamics is still unclear. Moreover, how much help such rules

provide in terms of developing systematic techniques for exploiting inefficiencies other than the wholesale use of the rules is also unclear.

The chartists themselves relate the formation of such rules to an understanding of what might be termed *market psychology*. They justify their techniques with arguments about the behaviour of investors. They do not claim to predict the behaviour of a market so much as understand the people who trade in the market. When a market reaches a previous high it is quite reasonable to imagine that many investors think of profit-taking, which in turn produces a "resistance" area [Ridl93]. If this type of analysis is true, or even statistically justified, then much clearer lines exist on which to base automated trading techniques, however this still leaves open any method for generating trading rules other than the use of a market expert.

2.4 An Adaptive Systems Approach

The above discussion raises two main difficulties associated with the construction of an automated approach to financial modelling. The first obstacle is theoretical: if markets are random then forecasting is impossible no matter what technique is employed. This view is essentially the Efficient Market Hypothesis, and the uncertainty arguments presented in sections 2.1, and 2.2. The second difficulty is practical: if financial modelling is possible, then how can predictive models be built, given that no formal theory exists on which to base them?

The theoretical point of view is clearly open and in all likelihood will remain open. As to the practical issue of how to construct predictive models one approach is offered by adaptive systems. From a technical perspective, adaptive/learning systems offer one approach to this problem, precisely because they are learning systems. To appreciate this, the following list of technical problems associated with financial/economic modelling is placed alongside the corresponding advantages an adaptive systems approach offers:

i) The first technical problem associated with the formation of a predictive financial model is the fact that there are no *a priori* theoretical models on which to base them.

Adaptive Systems Approach: Adaptive systems by their very nature do not require formal prerequisites, nor *a priori* models for the target system. They are designed to find their own representation of the problem instance and hence are not reliant on an existing theory on which to base their solution. Naturally, if no solution exists the system will fail. However, this does not prevent the system from being applied. This contrasts with the formal prerequisites demanded by traditional statistical approaches.

ii) How can determinants be selected in order to formulate a model?

Adaptive Systems Approach: An adaptive system can learn from examples to formulate a relationship between determinants. This does not guarantee success, however it does provide a flexible means for examining possible causal relationships between determinants and a target process. This contrasts with traditional statistical approaches, which rely on expert intervention and interpretation in order to construct models. No adaptive system will be completely free to

find relationships some boundary will always be defined by the number of determinants made available to the system.

iii) How can a model deal with non-linearity in the target process?

Adaptive Systems Approach: Adaptive systems are non-linear in that they have the ability to change and re-evaluate their knowledge on the basis of performance. This potential provides some way to deal with possible feed-back mechanisms thought to be prevalent within economic and financial systems.

iv) How can a model deal with uncertainty and changing structural determinants?

Adaptive Systems Approach: If the relationship between determinants changes over time, then adaptive systems are the only approach available in the absence of precise analytic descriptions. Only by adapting to the new situation can a system have the potential of maintaining any descriptive/predictive qualities. This is not to suggest that adaptive systems will successfully cope with all "regime" changes, on the contrary, no matter what system is used some *response time* will be necessary in which to adapt to the new situation (time in which the situation may have changed, and therefore the possibility of the system being forever out of phase with the target process is entirely credible).

iv) If the most successful financial/economic relationships rely on some fuzzy rationality, is it possible to incorporate this information in a model?

Adaptive Systems Approach: Adaptive techniques such as neural networks were developed with this sort of difficulty in mind – that is, to deal with decision problems on the basis of noisy and incomplete data [Tayl93] (for example in machine vision [MiPa69], [DuHa73],).

The above provides an idealised view of the advantages of an adaptive system approach. There are considerable technical problems associated with the application of adaptive systems within all domains, and specifically within finance, that are not reflected in the short descriptions of the advantages presented above. This point aside, the fact that adaptive systems offer a potential solution to some of the problems associated with financial modelling has not gone un-noticed. In the last four years there has been a rapid rise of interdisciplinary research combining time-series analysis with learning systems. In particular, considerable research effort has focused on the use of neural networks for financial time series analysis. Neural nets will play a central role in the adaptive system developed within this thesis for financial analysis. One of the main reasons for choosing neural networks as opposed to other adaptive methods is the success the technique has had in the area of financial modelling. In the next section we substantiate this comment by reviewing some of the progress that has been made in developing neural network solutions to financial and economic time series analysis problems.

2.5 Neural Nets and Financial Modelling

One of the first explicit uses of neural networks for time series analysis was in 1987 when Lapedes and Farber [LaFa87] demonstrated that feed-forward neural networks could be used for modelling deterministic chaos. The fact that a neural network could be used as a functional imitator in this form with little pre-knowledge of the application domain meant that a natural extension of this work was the use of neural networks for financial modelling. The idea of testing whether neural networks were capable of detecting underlying regularities in asset price movement was in 1988 with the first wave of economic time series applications. White [Whit88] used a feed-forward network to forecast IBM daily stock returns. The results were disappointing and did not provide evidence against the Efficient Market Hypothesis. However, it did highlight some of the practical problems involved in the design of neural network time series application. For example, it was reported that small nets could over-train on data sets of over a 1000 points. This emphasised the difficulty in deciding on the "correct" form of neural network training, and the importance of the correct network architecture.

More encouraging results were reported in [KAYT90]. The authors used neural networks to decide the buying and selling times for stocks on the Tokyo stock exchange. The authors applied techniques developed for phoneme recognition directly to find turning points in the price movements of selected assets. There are several interesting aspects to this early example: firstly they introduced methods for pre-treating the data, using moving averages and *log* compression. Secondly they used modular nets, making forecasts on several related series and then combining the results for their target prediction. The paper is short and does not provide full details of the level of returns achieved, and the test period was fairly short (2 years), however the authors reported that the system, know as TOPIC, delivered an excellent profit.

Dutta and Shekhar [DuSh88] applied neural networks to another aspect of market analysis by using a net to predict the ratings of corporate bonds. Here, rather than forecast the likely dynamics of a financial series, the net was used to rate the likelihood that a corporation will be able pay the promised coupon and par value on maturity of a bond. The so-called default risk represents the possibility that these commitments will not be paid. Generally the default risk of the most commonly traded bonds are estimated by specialist organisations such as S&P and Moody's. To evaluate a bond's rating a panel of experts assess various aspects of the issuing company. The precise methods employed by an agency to produce a rating is subject to commercial confidentiality and therefore unknown. However, certain aspects that are known to be assessed are factors that are hard to characterise, such as an institution's willingness to pay. Such factors have made bond rating hard from a traditional statistical analysis perspective [MoUt92]. Dutta and Shekhar reformulated the problem as one of classification based on known input-output cases, i.e., a set of factors known and the corresponding ratings produced. They compared the results obtained with a neural network and regression models, giving firm evidence that the neural network approach out-performed linear analysis.

Combinations of neural networks and other techniques have also been applied. Vergnes [Verg90] reports a series of experiments where neural networks are used to supplement an expert

system trading package. In this report the expert system approach is criticised as suffering from *frozen knowledge*, in that the system does not cope with regime changes. Moreover, despite the fact that new rules can be added, old rules can be rigid and inflexible to new events. In contrast, the author suggests that the added feature of a neural network provides continuous learning in that the weights within the network can be continuously adjusted, adapting to new conditions of the market, and consequently forgetting, if necessary, old rules.

Along these lines Abu-Mostafa [AbuM93] has introduced a technique for including hints, or previous knowledge, into network training. The approach incorporates expert knowledge to neural network training directly by training the network with standard gradient descent on both a training set and on the hint. For example, a simple hint is applied to currency exchange forecasting in which the hint states that if two networks are forecasting Dollar/Mark and Mark/Dollar exchange rates, both networks should produce opposite forecasts for any given period, i.e., if one network forecasts a rise in the Dollar compared to the Mark, the reciprocal network should forecast a fall in the Mark compared to the Dollar. Both networks can then be compared to not just network output, but also this reciprocal arrangement. In the experiments conducted this technique is shown to improve network generalisation as well the overall robustness of the network's performance.

Comparisons and reports on different styles of neural network training have also risen in the last five years. In [Scho90] a methodical comparison between the use of a perceptron, *adaline*, *madaline* and backpropagation networks for stock price prediction is carried out. Adaline and madaline are early perceptron like neural networks in which linear threshold functions are used as activation functions. The madaline structure has a layer of adaline units connected to a single madaline unit, which takes a majority vote rule for activation. The paper concludes that all models have some worth but that the backpropagation algorithm exhibits the best behaviour.

In [Refe92] a system for tactical asset allocation in Bond Markets is given. The system performs quantitative asset allocation between Bond Markets and US dollars to achieve returns in dollars, which is hopefully, in excess of all industry benchmarks. The seven markets considered are USA, Japan, UK, Germany, Canada, France and Australia. The interesting aspect of this application is the use of a modular approach to each market. Each local market is modelled with the aim of producing a local portfolio (that is local market plus US dollars) which outperforms a local benchmark. The results for the individual markets are then integrated in the global portfolio (full seven markets). There is no explicit forecast of exchange rate movement. Again good results are reported, and the system is reported to outperform a benchmark which is constructed from a combined portfolio (using the proportion of the global market capitalisation represented by each market).Another area of research that has been emphasised in the financial neural network community is that of network validation. Once more this relates to the design of the network, and on how best to achieve optimal training rules, architectures, and training parameters. Moody et al. [MoUt92] assess network performance in terms of *prediction risk*. This notion is intended to capture more precisely the idea of generalisation. It consists of multiple cross validation and the use of linear assessment criteria for the number of parameters in the model. Complexity is once

more penalised in the assessment of the network's performance. The techniques, are applied to bond rating and are shown to outperform linear regression analysis. Cross validation techniques in conjunction with suitable assessment criteria, are described as a vital component to the network's performance.

This theme is also present in Weigend et al. [WeHR90], where the authors introduce the use of a complexity term for the network size. It had long been realised that the network architecture is fundamental to the ability for the net to generalise outside the training set [BaHa89]. Weigend et al introduced an extra term to the standard gradient decent model in order to penalise complexity in the network's architecture. In this way the net is pruned as training proceeds. The justification for this being that simpler solutions should be preferred to more complex mappings. The argument as to how best find the optimal architecture, training rules, and activation functions is still very open and is itself the subject of intensive research. In a more recent work by Weigend et al [WeRH91b] the same weight elimination technique is applied to currency forecasting. They comment that the performance of the network is dependent on the weight elimination procedure, and give some comparisons between different architectures.

In summary, the last four years have seen a considerable rise in the use of neural networks in financial and economic applications. The above sites the main branches of this research effort. Other application not mentioned include marketing, advertising, sales statistics and business forecasting [FeKi95]. The consensus is that neural networks perform at least as well as traditional techniques, and in many instances significantly out-perform them [FCKL90].

2.5.1 Comparisons between Neural Nets and other Time Series methods

Underlying the use of neural networks for financial time series analysis is the idea that neural networks out-perform traditional modelling techniques. Over the last few years a number of empirical studies have been conducted to investigate this claim [HOCR92], many of which have been based on data from the *M-competition* [Makr82]. The M-competition set out to compare different forecasting techniques by running a competition. Makridakis et al gathered 111 real-world time series and presented them to various groups of forecasters, with the most recent values held back. The competitors were asked to make forecasts over the withheld data and the results were compared. Although this competition pre-dates neural network time series modelling, the experiment has since been re-run using neural networks on a number of occasions.

Study	Experiments	Conclusions
[HOC92]	111 Time Series	Neural nets superior to classical models
[SaEW90]	92 simulated 1 real series	Backpropagation best model
[ShPa90]	75 time series	Neural networks compatible to auto-box
[ShPa90b]	111 time series	Neural networks out-perform Box-Jenkins
[LaFa87]	3 chaotic time series	Neural networks better than regression models.
[TdAF90]	3 time series	Neural networks better long terms forecasting than Box-Jenkins
[FoCU91]	111 time series	Neural networks inferior to classical methods.

Table 2.5.1: Neural network forecasting comparisons.

Sharda and Patil [SaPa90] used 75 of the 111 series and found that neural networks performed as well as the automatic Box-Jenkins (auto-box) procedure. They noted problems with the remaining 36 times series. Foster, Collopy and Ungar [FoCU91] used neural networks on all 111 time series and reported that the neural networks were inferior to the best classical techniques. In a review paper of these results Hill et al [HOCR92] point out that this study failed to conform to a list of criteria set by the original panel of conference organisers for judging comparisons between techniques. This prompted Hill et al to re-run the experiments of the M-series, but this time carefully reproducing the circumstances under which the first M-competition was organised. Their results, summarised above, vindicate neural network techniques. They maintain that in most instances neural networks were usually better, and at worst were equivalent to classical techniques.

A more recent Makridakis' style competition has been run from the Sante Fe Institute and the results published in Weigend et al. [WeGe93]. Five time series were collected: a clean physics experiment (fluctuations in a far-infrared laser), physiological data from a patient with sleep apnea (heart rate, chest volume, blood oxygen concentration, and EEG state of the sleeping patient), high-frequency currency exchange (ten segments of 3000 points each of the exchange rate between the Swiss franc and the US dollar), a numerical sequence designed for the competition, astrophysical data from a variable star, and finally a fugue (JS Bach's final unfinished fugue from the Art of Fugue). After selecting the data series the participants were asked to:

- predict a withheld continuation of the series with respect to a given error measure,
- characterise the system (including aspects such the number of degrees of freedom, predictability, noise characteristics and the non-linearity of the system),
- infer a model of the governing equations, and
- describe the algorithms employed.

A fundamental shift in the experimental techniques was noted by the authors. They point out that for the Makridakis competition almost all techniques were linear, whereas ten years later and almost all were non-linear. neural nets were well represented and did very well in comparison to state space reconstruction techniques (the other dominant technique that was employed). In strict accordance with the rules some form of neural network approach *won* in each of the categories. The mention of strict ruling is important in that the authors applied additional experimentation to a number of the entries and found other methods (particularly state space reconstruction) as having desirable robustness qualities.

Weigend et al [WeHR90], [WeRH91] compared two styles of neural network, using sigmoid and radial basis functions, with a number of advanced statistical techniques for predicting sunspot activity. The sigmoid functions were found to produce far better results than the radial basis networks, and were also shown to out-perform a threshold auto-regression model, a bilinear model, a weighted linear predictor and were comparable with multivariate adaptive regression splines (MARS) [Frie91].

One factor that strongly supports the empirical evidence in favour of neural networks as compared to other modelling techniques is the fact that feed-forward neural networks have been shown to be universal functional approximators [see Chapter 3]. This fact ensures that in principle it is always possible to find a feed-forward network capable of approximating the functional behaviour of all other forms of modelling technique. In this sense neural networks should always be capable of matching the performance of other modelling styles, and that the real issue is not so much whether a neural network can out-perform a given technique but rather how easy it is to construct a good model. To some extent the power of neural network modelling can hinder the construction process in that a network is always capable of finding a spurious model that happens to match the training data. Such models do not guarantee good generalisation. This issue will be discussed in greater detail in the next chapter.

2.6 Genetic Algorithms in Finance

Neural nets are not the only adaptive technique that have been applied for financial modelling. One recent techniques that is finding increasing use in this area is Genetic Algorithms (genetic algorithms). The algorithm offers a means for interrogating large data sets and probing complex functional relationships, and has been rapidly established as a general-purpose optimisation tool. Before giving some of the financial applications of genetic algorithms it is worth describing the basic principles of the algorithm.

2.6.1 The Genetic Algorithm Search Technique

Genetic algorithms [Holl75], [Davi91], [Gold89] are a class of probabilistic search techniques based on biological evolution. The algorithm's search strategy is based on *biological evolution,* using a computationally simulated version of *survival of the fittest.* Figure 2.6.1 depicts the genetic algorithm evolutionary cycle. This provides the core search mechanism.

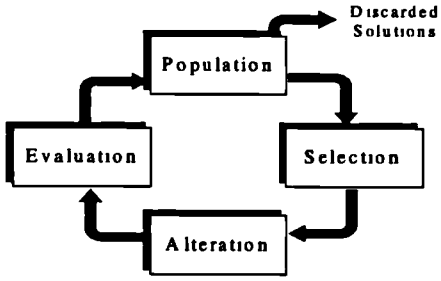


Figure 2.6.1: Genetic Algorithm Evolutionary Cycle

The algorithm mimics natural selection by repeatedly changing, or evolving, a population of candidate solutions in an attempt to find the optimal solution. The situation is akin to multiple scenario testing, each scenario being an *individual* within the population. The relative success of an individual is considered its *fitness*, and is used to selectively reproduce the most fit individuals to produce the next generation. Individuals represent knowledge through a collection of *chromosomes* each of which defines an aspect, or constraint, of the search space. The search is driven by the repeated interaction of the population with both itself (artificial mating, and a form of mutation) and the environment (through fitness evaluation and selection). By iterating this

process the population samples the space of potential solutions, and eventually may converge to the most fit.

Specifically, consider a population of N individuals x_i, each represented by a *chromosomal* string (or string) of L *allele* values. Allele refers to the specific value of a *gene* position on the string. A simple application of a genetic algorithm is in function optimisation. For example, consider the function f defined over a range of the real numbers $[a,b]$. Each $x_i \in [a,b]$ represents a candidate solution to the problem - $\max f(x); x \in [a,b]$. A simple representation for each x_i is a binary bit string, using the usual binary encoding. Here each gene position, or allele value, takes on either 0, or 1 depending on the value of x_i. The task is to maximise the output of f by searching the space $[a,b]$. In this case f is the *fitness* measure, or *fitness function*, and represents the environment in which candidate solutions are judged. The initial population is generally chosen at random. Once the population is initialised the genetic algorithm's evolutionary cycle can begin. The first stage of the genetic cycle is the evaluation of each of the population members. In the example above this equates to evaluating each $f(x_i)$ for all population members ($1 \le i \le N$). There then follows the repeated application of the biological operators. In the general case we have the following:

i) Selection: Selection is the process by which individuals survive from one generation to the next. A *selection scheme* is a means by which individuals are assigned a probability of surviving based on their relative fitness to the population as a whole. Individuals with high fitness should have a high probability of surviving. Individuals with low fitness should have a low probability of surviving.

ii) Crossover: This is a version of artificial mating. If two individuals have high fitness values, then the algorithm explores the possibility that a combination of their genes may produce an offspring with even higher fitness. Individuals with high fitness should have a high probability of mating. Individuals with low fitness should have a low probability of mating. Crossover represents a way of moving through the space of possible solutions based on the information gained from the existing solutions.

iii) Mutation: If crossover is seen as a way of moving through the space based on past information, mutation represents innovation. Mutation is extremely important, some forms of evolutionary algorithms rely on this operator as the only form of search (i.e., no crossover). In practice it is random adjustment in the individual's genetic structure (generally with a small probability).

The last two operators are often described in terms of *exploitation* of information encoded in good individuals (through crossover) and *exploration* of the search space (through mutation).

Having applied the biological operators the process is repeated until either the population converges (all members are the same) or some fixed control parameter is violated (such as set number of generations).

Figure 2.6.2 depicts a typical form of crossover and mutation defined for a binary encoding of the search space. Holland crossover, depicted above, picks a position m, $1 \le m \le L$ (where L is the

string length) at random and builds two offspring from two parents by swapping all bits in the positions $m \leq j \leq L$ on both strings. The central feature of Holland's genetic algorithm is the use of crossover. An intuitive idea behind the inclusion of the crossover operator would be that if two parents with above average fitness mate then it is possible that the offspring will combine complementary substructures from both parents, and therefore increase fitness. It provides a means of information exchange between elements involved in a parallel search. This, together with selection, offers a simple way in which to signal areas of higher fitness, so that the algorithm's search can be focused in a natural way. Mutation for binary encodings is generally defined as a small probability that a bit value changes.

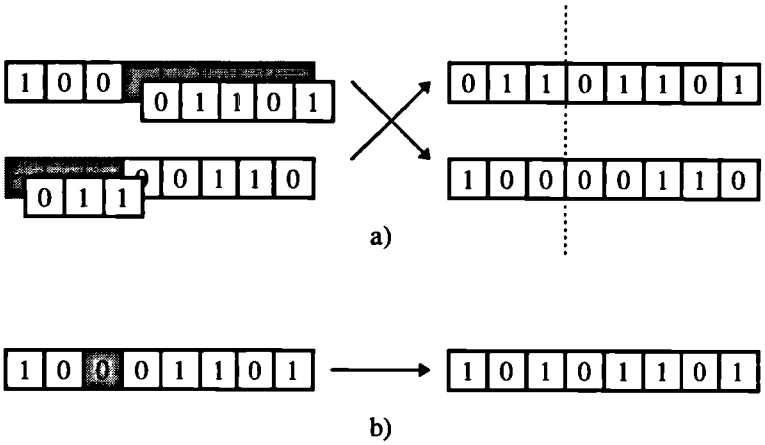


Figure 2.6.2: Genetic Operators a) Crossover, b) Mutation

To complete the terminology, a set of chromosomes of an individual is referred to as its *genotype*, which defines a *phenotype* with a certain fitness. A genetic algorithm is a parallel search algorithm with centralised control. The centralisation comes from the selection regime. The fact that it is a parallel search relates to the fact that there is a population of candidate solutions. This form of parallelism should not be confused with either parallelisation (in terms of the actual implementation of the algorithm), nor with *intrinsic parallelism* (which relates to one of the explanations as to why genetic algorithms are effective). These aspects will be discussed in Chapter 4 in more detail.

2.6.2 Applications of Genetic Algorithms

The last four years has seen a rapid expansion in the commercial exploitation of genetic algorithms. To date they have been used in portfolio optimisation, bankruptcy prediction, financial forecasting, fraud detection and scheduling (notably the Olympic Games) [StHK94], [Gold89], [PoST91], [Dav89], [KiFe95]. In Europe, the ESPRIT III project PAPAGENA was the largest pan-European investment into the research, exploration and commercial development of genetic algorithms The two-year project demonstrated the potential of genetic algorithm technology in a broad set of real-world applications. These included protein folding, credit scoring, direct marketing, insurance risk assessment, economic modelling and hand-written character recognition [DeKi94]. genetic algorithms have also been used for assessing insurance risk [Hugh90], portfolio optimisation [Nobl90] and financial time series analysis [GoFe94]. In

addition genetic algorithms have been used in a number of studies attempting to model behaviour within speculative markets in forecasting company profits and calculation of budget models. In the US, First Quadrant, an investment firm in Pasadena uses genetic algorithms to help manage $5 billion worth of investments. It started using the technique in 1993 and claims to have made substantial profits. Currently the company uses genetic algorithms to govern tactical asset management in 17 different countries [Kier94]. Many other investment houses both in the US and Europe are rumoured to be using the technique, and a recent book dedicated to genetic algorithms for financial trading would suggest there is some level of awareness and interest in applying this technique [Baue94].

2.7 Summary

This chapter has outlined the technical and theoretical difficulties associated with financial time series analysis. It has been shown how some of the technical problems associated with financial modelling can be dealt with, at least in principle, by adopting an adaptive system approach. It was shown that neural networks offer many attractive features when attempting to model poorly understood problems domains, and that over the last few years they have established themselves as credible financial modelling techniques. At present neural networks are the most widely applied adaptive technique within the financial domain, however genetic algorithms are also found in an increasing number of applications, and certain characteristics of the genetic algorithm search process is well suited to the financial domain.

In terms of this thesis we are interested in investigating the possibility of designing an automated financial time series analysis system. What this chapter has attempted to establish is the fact that neural networks are a good first choice in terms of a technique for financial modelling. Moreover, the widespread usage of neural networks in financial modelling implies that any methods, or techniques, that are introduced which make this task easier will be of general interest, and of genuine use in both the financial and research communities. Having made this decision, the next most immediate task is to assess the technical problems associated with developing an automated approach to the application of neural network modelling. We start this process in the next chapter, where we take a detailed look at the neural network modelling process, and make explicit the technical problems associated with a successful neural network application.

Chapter 3

Feed-Forward Neural Network Modelling

The purpose of this chapter is to demonstrate and explore the technical difficulties associated with an automated application of a feed-forward Neural Network (NN).

3.1 Neural Net Search

From the applications surveyed in Chapter 2 there is no doubt that neural nets offer a great deal of scope for providing the core of an automated financial time series analysis system. However, all of the applications reviewed in Chapter 2 were the result of careful experimentation on the part of a human designer, and the learning, or training, aspect of the neural network has simply been a single, and possibly last, phase in the development process. This in effect has far more in common with other applied statistical methods than machine learning, and therefore leaves many questions open as to how to achieve an automated approach.

The problems faced centre on the many parameters associated with the *design* of a neural network application. These can range from the topology, the learning method, the choice of activation function, the number of training cycles, and even the scaling, and pre-treatment of data, all of which directly affect the likelihood of the network finding a good solution. Furthermore, the sensitivity of a neural network to its parameter settings, as with all forms of non-linear techniques, means that there are real dangers in producing both false positive and false negative results. As mentioned in Chapter 2, methods for dealing with these issues are, at present, very open, and consequently a human designer would still seem the best choice. This implies that any attempt to design an automated neural network system must ensure that a comprehensive systematic decision process is in place to deal with the parameterisation issues. To start this process we take a closer look at a generalised form of Multi-Layer Perceptron (MLP) training in terms of time series analysis, and explore the design choices an automated system will have to make.

3.2 MLP Training: The Model

A univariate time series problem consists of a time-dependent process X generating a sequence of observations according to the following signal plus noise relationship[1],

$$x_t = g(x_{t-1}, x_{t-2} \ldots x_{t\ n}), \qquad 3.1$$

x_t is the current response (dependent variable), $x_{t-1}, x_{t-2} \ldots x_{t-n}$ are past values of the series, ε_t is observational noise and g is the target function to be estimated. If g is a scalar multiplier then the above becomes a classical auto-regressive model of order n. In most real-world problems few *a priori* assumptions can be made about the functional form of g. Since a parametric class of

[1]For the present we ignore the affects of noise.

functions is not usually known, one resorts to non-parametric regression approaches, such as MLP, where an estimate $\hat{g} = f$ for g is constructed from a large class of functions F. In the case of MLP, F would be defined, or bounded, by the architecture of the network, or networks under consideration, and the task of approximating g is the problem of parameterising F in order to construct $\hat{g} = f \in F$. A typical class of MLP using one hidden layer and one output node can be described by the following:

$$\hat{x}_t = A\left(\theta_0 + \sum_{i=1}^{h} w_i' A\left(\sum_{j=1}^{n} w_{i,j} x_{t-j} + \theta_i\right)\right), \qquad 3.2$$

here, h denotes the number of hidden units in the net, n the number of input units, $w_{i,j}$ the weights connecting input unit j with hidden node i and θ_i acts as a bias for node i. w_i' is the weights connecting the hidden node i to the output node, with θ_0 acting as its bias. The activation function A (for this thesis) will be one of a class of sigmoid functions described by,

$$A(y) = K + \frac{\alpha}{1 + e^{\beta y}}, \qquad 3.3$$

which for suitable choices of the constant terms K, α, β give the typical sigmoid $(K = 0, \alpha = 1, \beta = -1)$ and hyperbolic tangent $(K = 1, \alpha = -1, \beta = 2)$ *mapping* functions. Note that all of the above can easily be extended to the multivariate case with multiple outputs. All that is required is additional input/output nodes with appropriate extensions to the summations in equation 3.2.

Having defined the output of the net the training phase consists of adjusting the weight vectors to minimise a cost function, $E = C(\hat{x}_t, x_t)$, where E is the error of the net as defined by the cost C. The cost function is variously defined [Mood92] but overwhelmingly it consists of a single distance measure (e.g., squared difference) between the network output and the desired output pattern. In the general case the parameterisation of equation 3.2 is affected by some form of gradient descent over the weight landscape in order to minimise the cost function for the complete training set. For time series analysis this has the effect of windowing through the data, creating input/output instances according to the order of the model given in equation 3.2. This is depicted in Figure 3.2.1.

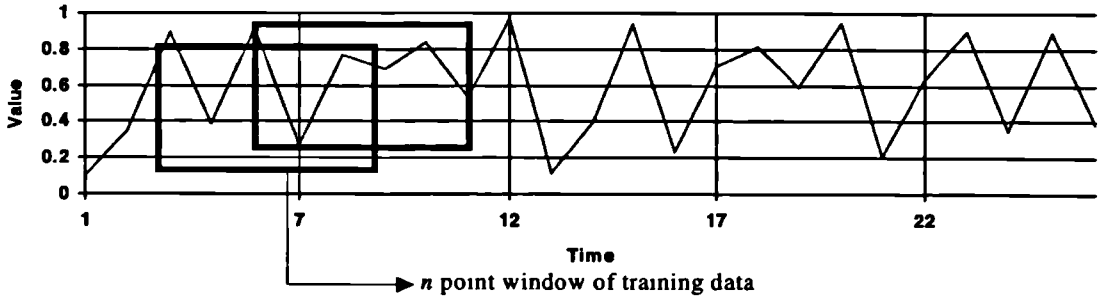


Figure 3.2.1: Network Time Series Training.

In order to minimise the cost function over the training set we have the following;

$$\frac{\partial E}{\partial w_i'} = \frac{\partial E}{\partial \hat{x}_t} \cdot \frac{\partial \hat{x}_t}{\partial w_i'}, \qquad 3.4$$

for the hidden to output node's weights, and letting $y_i = A(\sum_j^n w_{i,j} x_{i-j} + \theta_i)$ in equation 3.2 we have,

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial \hat{x}_i} \cdot \frac{\partial \hat{x}_i}{\partial y_i} \cdot \frac{\partial y_i}{\partial w_{i,j}}, \qquad 3.5$$

for the input to hidden weights. To finish the derivation of gradient descent over the weight space the weight update rule is given by,

$$\Delta w = -\lambda \frac{\partial E}{\partial w}, \qquad 3.6$$

where λ is a *learning rate* which dictates the size of movement each weight is changed in respect to its error derivative. There are several ways in which the actual movement over the error surface is controlled, in particular the way in which λ is set. Second-order methods involve calculating the second derivative of the error with respect to a weight and using this in conjunction with the first derivative to accelerate the learning process [Watr87], [Park87], [BeLe88]. However, using the second derivative means more complex weight update rules must be applied to avoid situations where its value is negative (this can lead to hill-climbing). A less complex method, which is generally accepted as being as fast, is a learning update rule, or dynamic learning rate [Son93].

Dynamic learning adjusts the value of λ associated with each weight dynamically so that the step size associated with each weight change is adjusted during training. It proceeds by evaluating the sign associated with each component of the gradient vector for each iteration (or training cycle). If a given component presents the same sign in two successive iterations the corresponding learning rate can be increased. On the other hand when the sign of a given component changes this means the learning procedure has passed over the minimum along that direction, and the learning rate must be decreased in order to reach the minimum. λ can either be adjusted for each individual weight or by taking into account aggregate weight changes [Jaco88].

3.3 MLP: Model Parameters

The MLP detailed above includes a considerable number of parameters, all of which have various levels of tolerance in order to generate a successful neural network model. In Table 3.3.1 a list of parameter choices is set out. This represents the minimum number of design decisions an automated system will have to make in order to apply an MLP to a given learning problem.

Clearly Table 3.3.1 represents an optimisation problem in its own right. In order to design an automated system for neural network applications some way of dealing with each of the parameter choices set out in Table 3.3.1 will have to be found. To start this process we assess the relative importance of each of the entries in 3.3.1, and where possible either fix parameters, or suggest means for an automated decision process.

Parameter Choice	Options Available and Problems Raised
1. Data Selection	For a given target process many possible causal effects may be relevant in the construction of a model. An automated system will require some method for determining how secondary series can be inferred for a non-linear generating process.
2. Data Pre-processing	What forms of data treatment such as scaling, smoothing should be applied to training data?
3. Forecast Horizon	How should an optimal forecast horizon be chosen?
4. Training data	Having selected and treated data, how much should data be segmented for network training?
5. Order of Model	What should the number of lags, or number of input nodes, for the NN be?
6. Architecture	What should the number of hidden nodes be?
7. Activation Function	Which is the best activation function in any given instance and what bias does it introduce?
8. Learning Rule	What rule should be used to control the weight update during training?
9. Learning Style	What style of network training should be used i.e., batch versus on-line?
10. Convergence	Since the learning method is gradient descent there are many local minima that the model can get stuck in, therefore how is training to be stopped?
11. Cost Function	How should the cost function be defined? What variation to the results does it imply?
12. Generalisation	Once all of the above have been completed how is the network to be validated? What confidence criteria can be placed on the network's likely performance in terms of generalisation?

Table 3.3.1: MLP training parameters.

3.4 The Data

The first 5 entries in Table 3.3.1 relate to the data and to ways of analysing and manipulating the data in preparation for neural network modelling. Model formulation is very open, and according to Chatfield has been neglected in the classical time series analysis literature [Chat89]. One of the reasons for this is the fact that in order to pre-treat data in an effective manner, it pre-supposes the existence of a theory (or model) of that data. For example, if the model is assumed to be linear then there are many techniques that will aid the construction of a linear model. For example, covariance analysis can be used to determine the number of lags, or order of the model, it can also be used to establish related time series. If however, the model is assumed to be non-linear, which is the case in most financial modelling, then the only effective pre-treatment of the data will be treatment that goes some way in actually formulating the target model itself. Hamming [Hamm86] gives a good example of this when he points out that the entropy of a series of pseudo-random numbers will be high, indicating randomness, whereas the entropy of the series taking into account the generating process will be zero. This implies that data pre-treatment must make some reference to the hypothesis space of possible models offered by the modelling process, which in this instance refers to the class of functions offered by a trained neural network. At present it is down to the model builder to infer the possibility that a secondary data series is influential in a non-linear manner for a target process. We shall return to the question of how to automate this process in Chapter 7, once the target financial series is introduced.

In almost all non-trivial applications of neural networks some form of data pre-processing is necessary. Regardless of the form this takes, safeguards must be present to ensure that pre-treatment does not corrupt the modelling process. An example of the problems that can be

encountered is shown in Figure 3.4.1. Here a 6-4-1 totally connected MLP has been trained on the values for the symmetrically bounded ramp function, in the range (-10,+10). The generating process is given below,

$$f(x)=\begin{cases}-10, \text{ if } x<-10,\\ x, \text{ if } -10\le x\le 10,\\ 10, \text{ if } x>10.\end{cases} \qquad 3.7$$

Using Hyperbolic Tangent activation functions at all nodes (including output nodes), the net has a functional range (-1,+1). This implies that the data must be scaled to lie within this range. Three networks were trained for exactly the same number of training cycles, with exactly the same training parameters, the only factor that was changed was the scaling of the training set for each of the nets. Using different scalings of the training set Figure 3.4.1 shows how the network's forecast is forced to dip as the output values approach the limit of the activation function's range.

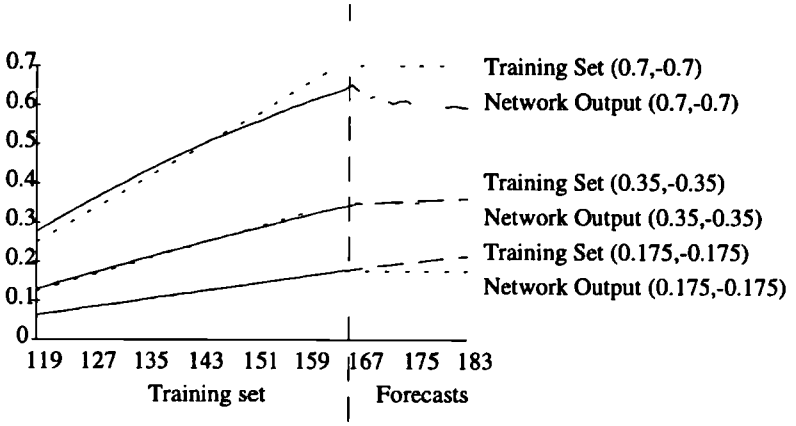


Figure: 3.4.1: Network Forecast on the Symmetric Ramp

This effect shown in 3.4.1 can essentially be produced at will by selecting appropriate scalings of the training set. This type of turning point can produce both false positive as well as false negative results. This makes it imperative that the validation set is not included in any form of pre-processing of the data. It also suggests that a single control parameter (the scaling) can directly effect the results produced by the network, and that if control parameters are adjusted in line with a network's validation performance, the validation set must be seen as corrupted in terms of a fair experiment. In practice the above implies the need for a much deeper understanding of the likely models offered by the neural network training process. We have already mentioned over-training, and incorrect generalisation and clearly want to avoid this in an automated system. The specific requirements of data pre-processing used within this thesis will be discussed in Chapter 7 when the target financial series is introduced. In the following sections we investigate each of the MLP training parameters.

3.5 MLP: Training Parameters

Entries 5 to 8 in Table 3.3.1 relate to the parameterisation and solution space offered by an MLP. The following sections investigate these parameters and the effects they have on the solutions preferred by a particular neural network. We start with the architecture.

3.5.1 Architecture

The prime concern in a system that is trained by examples is how well it generalises out-of-sample. If the system simply memorises the training set then it is likely that it will perform well in-sample and badly out-of-sample. The architecture of a MLP is important in this respect as it dictates the number of free parameters available during training. If there are a large number of free parameters and a small number of training examples then it should follow that the network is more prone to over-fitting.

We can substantiate this intuitive argument with a simple demonstration relating to the ability for MLPs to act as universal functional approximators. MLPs have been investigated in terms of universal approximators on many occasions [Cybe89], [Funa89], [Horn91], [HoSW89], [SaAn91]. For most of these results the researchers have relied on measure theoretic properties of neural networks and have in all cases produced existence proofs.

A consequence of the rather theoretical nature of these results is that few practical guidelines have been offered for the applied community, in the sense of explicit topologies, or even methods for interpreting the nature of the trained neural network's functionality.

Below we present a novel demonstration that for any set of N input/output pairs there is an MLP using a single hidden layer of N-1 nodes, and sigmoid activation functions, that can make the desired map with arbitrary precision.

The easiest way in which to see how an MLP can act as a look-up-table is to note the properties of the activation function (we restrict the following to the $1/(1+e^{x})$ case, however all results carry over for all of the activation functions represented by equation 3.3):

$$\frac{1}{1+e^{m}} = \begin{cases} 0, \text{ for } m \to \infty, \\ 1/2, \text{ for } m = 0, \\ 1, \text{ for } m \to -\infty. \end{cases} \tag{3.8}$$

If the MLP is restricted to having one hidden layer and linear output nodes then for input-output pairs $\{x_i, y_i : 0 \le i \le N\}$, with $x_i < x_{i+1}$, and $x_i \in [0,1]$, we can define the i'th hidden node activation as follows:

$$\frac{1}{1+e^{m(x_i - x)}} \cong \begin{cases} 0, \text{ for } x > x_i, \\ 1/2, \text{ for } x = x_i, \\ 1, \text{ for } x < x_i. \end{cases} \tag{3.9}$$

for $m \to \infty$. That is, we can take the weight for input node to the i'th hidden node as $w_i = -m$, and the bias for this node as $\theta_i = mx_i$, where x_i is the i'th input pattern. It follows that by defining the input to hidden weights in this manner we achieve the following activation for input x_i,

$$A[x_i] \cong [1, \ 1, \ \cdots \ 1/2, 0, \ 0, \ 0], \tag{3.10}$$

where the i'th node gives the output 1/2. Therefore for the full set of input patterns we have,

$$A\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix} \cong \begin{bmatrix} 1, & 0, & \cdots & & & \cdots & 0 \\ 1, & 1/2, & 0, & \cdots & & \cdots & 0 \\ \vdots & & \ddots & & & & \vdots \\ 1, & \cdots & 1, & 1/2, & 0, & \cdots & 0 \\ \vdots & & & & \ddots & & \vdots \\ 1, & \cdots & & \cdots & 1, & 1/2, & 0 \\ 1, & \cdots & & \cdots & 1, & 1, & 1/2 \end{bmatrix}, \tag{3.11}$$

where for x_0 we define the bias to the output node as 1. Having defined the activation matrix for the hidden nodes over all input patterns we can then solve for the appropriate output weights. We require,

$$y_i = \sum_{j=0}^{h} w'_j A(x_i), \tag{3.12}$$

for which we get,

$$w'_j = \begin{cases} y_0, \text{ for } j = 0, \\ 2y_j - 2\sum_{i=0}^{j-1} w'_i, \text{ for } j \geq 1, \end{cases} \tag{3.13}$$

which gives,

$$w'_j = \begin{cases} y_0, \text{ for } j = 0, \\ 2y_j + 4\sum_{i=1}^{j-1} (-1)^{i-j} y_i + (-1)^j 2y_0, \text{ for } j \geq 1. \end{cases} \tag{3.14}$$

The above provides a map between input and output pairs, with the network acting as no more than a look-up table, with a precision dictated by the size of m. The above gives some insight into the actual mechanism by which an MLP may find a mapping. A possible extension of this result (not derived here) to include input/output cases not in the training set could follow a variation of the Weierstrass Approximation Theorem [Simo63] for polynomial approximations for bounded continuous real valued mappings from [0,1] to the real line. If for each node i of the hidden layer we take an activation for a given input $x \in [0,1]$ as follows:

$$A(x) = \frac{1}{1 + e^{m\left(\frac{i}{n} - x\right)}}, \tag{3.15}$$

where n is the total number of hidden nodes. The above implies that for sufficiently large values of n and m a fixed error bound should be achievable. Completeness is not the aim in presenting this demonstration, instead it is to show how the network can memorise a target function by using the hidden nodes to map evenly spaced input/output pairs over a training set. What the above shows is that the network topology can facilitate arbitrary mappings between input-output instances and that in order to achieve good generalisation careful consideration of the network architecture is required. In order to determine the precise, or even empirical, relationship between the architecture of a network and the likely generalisation some means for determining a *correct*

architecture is required. That is to say, unless we can test various architectures against a known solution the relationship between the network's results and different architectures will remain vague. In Chapter 5 we come back to this issue. In Chapter 5 we introduce a method for testing architectures against known solutions and use this technique as a means for developing an automated network architecture selection procedure. In this chapter we continue the general discussion of training parameters and their effect on network performance.

3.5.2 Activation Function

It has been shown [Horn91] that an MLP using an *arbitrary* bounded non-constant activation function is capable of universal approximation. However in practice the most common form of activation function used is one of the family of sigmoids given in equation 3.3. The activation function clearly affects the weight-to-error landscape and might suggest that more work should be done in how to choose an appropriate activation function given a problem instance. Some work in this area has been done on sigmoids and has shown that in most cases the Hyperbolic Tangent yields faster convergence (more generally this result holds for symmetric activation functions [LeCu89]). This has subsequently been supported by several large-scale empirical tests [King94]. For these reasons in this thesis the Hyperbolic Tangent is used for the hidden layer activation functions.

Another issue relating to the choice of activation function is the method of weight initialisation. The rule of thumb most researchers advocate is that the weights should correspond to the linear portion of the activation function. The reason for this is that if the activation function is saturated the gradient approaches zero and therefore produces very small weight changes in the gradient descent process. For this reason it may also be necessary to take into account the range of the training set so as to guarantee that the initial weights maximise the weight adjustment procedure during the early part of training.

3.5.3 Learning Rules, Batch and On-Line Training

The learning rule amounts to the way in which actual weight changes are effected. As mentioned in 3.1 there is a general consensus that dynamic learning rates are at least as effective as second-order methods [Jaco88]. As to the precise setting of learning rate multipliers there is little formal guidance. Most neural network researchers apply some heuristic based on experience. This usually means that a maximum learning rate is set in order to avoid too large movements over the weight space, and a corresponding minimum learning rate to avoid complete stagnation. There have also been results which have used secondary optimisation methods, such as genetic algorithms, as a means for setting learning multipliers [WhHa89]. These techniques will be discussed in Chapter 5.

Another question that is often raised about learning rates is whether the rate should be adjusted locally on a weight-by-weight basis, or globally for all weights. The arguments [Jaco88] mirror the discussions as to whether weight changes should be made after the presentation of each input/output pair (on-line training) or in accordance to the aggregate weight move given by the whole training set (batch training). Again at present, there is no formal basis for preferring

one method to another, and each appears to be effective in some circumstances. However, batch training has the potential to be faster in that in some circumstances it can avoid oscillations caused by the order in which patterns are presented [Hint87]. Local learning rates are also considered faster than global weight multipliers. This may be caused by the network being far more prone to finding local minima. In this thesis we use both batch training and locally adjusted dynamic learning rates.

Another common observation relating to dynamic learning rates is the use of an activation function at the output node [Mast93]. It has been suggested that the use of the activation function at the output layer helps to stabilise the performance of the network, particularly when making iterated forecasts. One of the reasons for this is the fact that large values are always dampened by the mapping function and therefore restrict some of the extremes of network output. However, it can be argued that if the network has approximated the *correct* mapping then iterated forecasts using linear output nodes are justified. Furthermore, linear output nodes avoids some of the scaling problems mentioned in 3.4. For this thesis the output nodes will test both linear and hyperbolic activation functions, with an automated decision process making the final choice for a given application.

3.6 Network Performance

The final entries in Table 3.3.1 relate to the performance of the trained neural network. The large number of parameters that are available to network training means that the validation phase will inevitably affect the choice of parameter settings during learning, that is to say, if the network has been trained and it performs badly (by whatever criteria) the most obvious course of action is to adjust parameters and re-train. The implications of this are discussed below.

3.6.1 Convergence

Network training proceeds by adjusting the network's response to each input pattern until the cost function is reduced to some pre-set tolerance level, or the number of pattern presentations violates some pre-set bound. In the first instance the network is said to have *converged.* As there is an upper and lower bound on the step size made by network during training, it means each weight is always adjusted by some finite amount. This implies that absolute convergence (with the cost function equal to zero) can never actually occur, which in turn implies some pre-set tolerance or limit on the training cycles must be made. There are several points raised by this, the first and most obvious is how should training be stopped?

When to stop training calls into question the very nature of the actual learning process itself. Cost function minimisation can clearly be met by many functions other than the desired mapping between input and output (this includes the trivial look-up mapping given in 3.5.1). This further implies that the real issue is not so much cost function minimisation, but the level of generalisation achieved by the trained network. This issue can be illustrated by the symmetric ramp example given in 3.4. As was shown for pre-treatment of the data (in 3.4 scaling) the same effect can be produced by varying the number of training cycles on the fixed data set. An identical set of results to those given in Figure 3.4.1 can be achieved by holding the scaling fixed

and varying the number of training cycles. This effect and the one given in 3.4 serve to highlight the fact that cost function minimisation is not the ultimate aim of network training, whereas generalisation is. Generalisation is clearly the key issue in terms of the learning process, and since the cost function is not a measure of generalisation it can be argued that convergence criteria are somewhat arbitrary in terms of what the trained network is intended for [King93]. This is important – it means that the network designer has no direct, or analytic, means for setting the network's free parameters to achieve good generalisation.

Traditionally the method employed to explore a network's generalisation ability has been the use of a validation set. The validation set refers to a portion of the training set that is withheld from the training data and is used as a measure of the network's out-of-sample performance. However, Figure 3.4.1 suggests that unless rigorous methods for the validation procedure are employed, over-fitting can quite easily extend to the validation set. That is to say, a network can perform badly on a validation set for many reasons other than the target series is random. Most researchers would adjust network parameters and re-train, possibly many times, before concluding that a network solution was impossible. In this way the validation set moves from being out-of-sample to in-sample and the experiment, or the performance metric, is corrupted. Ways in which to tackle this problem are clearly at the heart of an automated approach to network design and training. The reason for this is that an automated system will, by definition, make explicit relationships between the validation procedure and the network design. This will occur naturally as part of the system's network design phase. The effect of this will be to directly increase the complexity of the whole process, and therefore provide ample opportunity for the complete system to over-train. A good example of this is given in Figure 3.6.1 below:

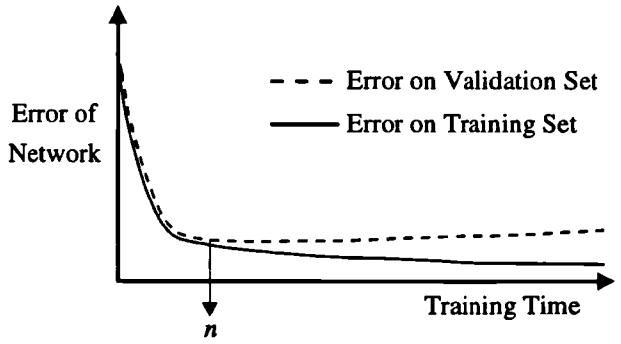


Figure 3.6.1: Network Training Mean Square Error Profiles.

Figure 3.6.1 shows a common profile for the in-sample training error and the out-of-sample validation error as the number of training cycles increases for a network under training. As can be seen the validation error gradually decreases as training proceeds until at some point, n, its value sharply increases, and the network would be judged to be over-trained. On the basis of the effect shown in 3.6.1 some researchers have suggested that the best way in which to deal with over-training is to simply stop training when the validation error starts to rise [Smit93]. However, this is a naive approach, and in reality represents little more than direct training on the validation set. The fact that neural networks are universal approximators, means that for any finite data set and a network of sufficient complexity we can always find a spurious mapping that matches input to

output. If the validation procedure consists of a single set of data held back from training, which is used as a control parameter for network training, the results obtained are corrupted, and in general will not produce good generalisation when used on genuinely out-of-sample data. The best demonstration of this problem is given with the following example. Figure 3.6.2 shows the output of a network trained on random data (using UK National Lottery Numbers). The iterated forecast is extremely good and was made over data that was withheld from the training process. However, the forecast horizon was used as a validation set during training, and the results during training on this set were used to set the number of training cycles.

In the next section we discuss some of the methods that have been developed to aid generalisation, as well as sketch out the techniques that have been introduced for this thesis.

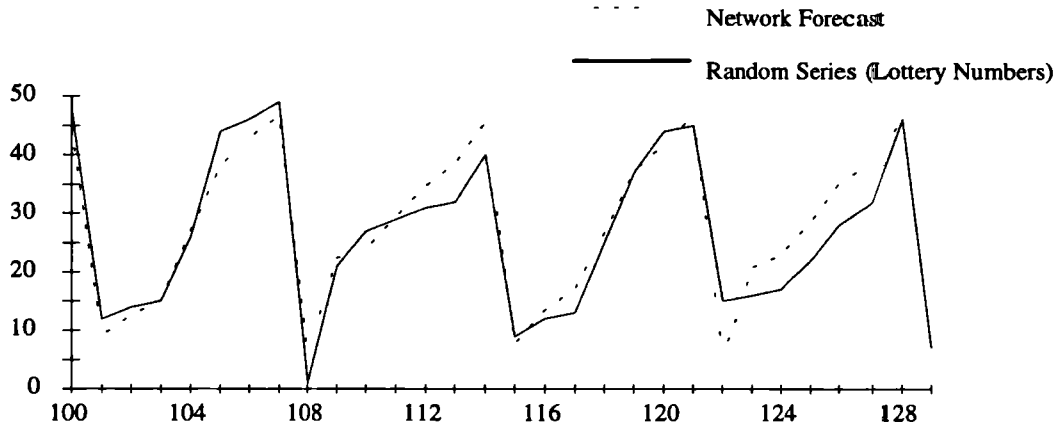


Figure 3.6.2: Network Forecast for a Random Series (Lottery Numbers).

3.6.2 Network Validation and Generalisation

Over-fitting and generalisation are always going to be a problem for real-world data, especially in circumstances where the data has little in the way of a clearly defined signal. This is particularly true for financial applications where the target series may well be random, or at least heavily contaminated with noise. The lack of *a priori* predictive models makes it very hard to construct metrics from which to judge the network's performance. Researchers who have devised methods for dealing with these issues, have generally applied some form of Occam's razor [WeHR90], [BEHW89]. Occam's razor is the principle that states that unnecessarily complex models *should not* be preferred to simpler ones. However, this is a heuristic, and more complex models always fit the data better, and in some instances the maximum likelihood models that include high levels of complexity can be the best model [Wolp93].

Broadly speaking neural network researchers (and for that matter, the learning theory community) have adopted three main approaches to the above dilemma:

i) Minimise Complexity - These techniques make direct assumptions about the target model. In practice the most common method is to apply a form of Occam's razor, where the network is penalised for *excess* complexity. This usually takes the form of regularisers introduced during neural network training (these techniques will be discussed in some detail in Chapter 5 when we discuss an automated method for architecture selection).

ii) Analysis of the Model - These techniques try to limit the *a priori* assumptions about the target model, and concentrate more on the underlying characteristics of the learning system. One approach is to conduct a detailed analysis of the hypothesis space offered by the learning system [Vali84]. In general a fixed learning system will be characterised by the class of possible relationships, or functions, it can learn. For example, given a fixed network architecture with fixed activation functions, the network bounds some class of functions F that can be realised by the network (i.e., by adjusting the weights). A learning problem can then be framed as follows: given a finite set of examples S of a specific function $f \in F$, the network's task is to use information gained from S in order to identify f. Network training is therefore the procedure by which a hypothesis $\hat{f}$ is made for f. If we further assume that the examples used to train the system are produced according to a fixed probability distribution, and we have detailed knowledge of F, we can make precise statements regarding the likelihood that $f = \hat{f}$. For example, the error of a hypothesis can be defined as the probability that $\hat{f} \neq f$ for new randomly chosen example of f, assuming that the new example is generated via the same probability distribution that produced S. Since there are only a finite number of training examples two sources of error arise: firstly, the training set is unrepresentative of f; and secondly insufficient examples were presented for the learning system.

Given the assumptions made above, the likelihood of both of these errors can be estimated and minimised for a specific learning system by controlling the number of samples required in the training set. This can then provide a probabilistic confidence measure that the learning system has identified the correct model. The most common form of this uses the Vapnik-Chervonikis (VC-) dimension, which can be used as a formal measure of the capacity of a learning system's hypothesis space [VaCh71]. This is usually used in conjunction with Probably Almost Correct (PAC) learning which can use the VC-dimension as a means of bounding sample sizes required to teach the system the correct mapping, to a pre-defined confidence limit [Vali84], [BEHW89], [Haus92].

There are several difficulties in using the above for neural networks and real-world applications. Firstly, we must assume that the target function is in the hypothesis space of the learning model, secondly there is a fixed probability distribution responsible for generating examples and thirdly we require a measure for the capacity of a specific neural network. The second requirement is relatively benign — most statistical modelling techniques rely on such assumptions (stationarity, for example). The difficulty arises in the first and third. In order to ensure a neural network contains the target mapping, a large network is preferable. However, this implies that the capacity, $|F|$, is also increased which in turn implies that for a finite training set S there will exist many $f_i \in F$ such that $f_i(x) \cong f(x)$ for $x \in S$ but $f_i(x) \neq f(x)$ for $x \notin S$. This last condition relates to the richness of the hypothesis space offered by a neural network and can imply very large training sets, too large for most practical circumstances. Research in the this area is still active, and to date a practical bound on the VC-dimension for continuous MLP style neural networks is still open.

iii) Validation Measures - A final approach attempts to minimise *a priori* assumptions about the target model by adopting strict statistical methods for the validation process. Here the question of model formation is ignored but a set of rigorous validation procedures are introduced. This is a vast area, and includes all of the usual classical statistical methods.

However, certain general procedures of validation have been adopted by the neural network community. These include multiple cross-validation, akin to the leave-one-out or multiple jack-knife techniques of statistical analysis [Mood92], [Chat89]. Cross-validation is a sample re-use scheme, in which the training set is divided into multiple disjoint data sets. The neural network is trained on one section of data and tested on another. Moody has used this method to define what is called the network's "prediction risk". This is a measure of the expected performance of the network on future data [MoUt92]. This statistic is estimated via a process of multiple cross-validation. The key point here is that the prediction risk can only be estimated, and to do this requires the introduction of certain assumptions about both the target function, the modelling method, the data sampled, and the sampling process. This is true of other statistical techniques that have been introduced to measure a network's performance, such as the Network Information Criteria (NIC) which is based on the Akaike's Information Criteria (AIC) [Chat89], which in turn is a generalisation of the Akaike's Final Prediction Error (FPE) (these last two examples are methods developed for choosing between linear models).The basic idea is to devise a method for selecting a model based on its performance on a validation set, as opposed to an in-sample statistic. In the last case, the FPE selects the model on the basis of the smallest mean square error of a one-step-ahead forecast (as in normal neural network time series training). The AIC generalises this to take into account the complexity of the model, and the NIC generalises this to take into account the non-linearity of neural networks.

A final method for validation and selection that is widely used for neural networks is a Bayesian approach. This is an alternative to the information criteria mentioned above and is based on classical Bayesian analysis. As in the techniques mentioned above, the problem of learning, or network training, is reformulated as a statistical sampling problem. One advantage of the Bayesian method is that the prediction for a test case is based on all possible values for the network parameters, weighted by their probability on the basis of the training set [MacK92], [Neal92]. This therefore avoids the question of over-training in that Bayesian training is not an optimisation problem as such, but an integration problem. In practice, the chief difficulties that arise with this method relate to the assumptions that are required in order to estimate the probability distributions of the weights, and the errors produced by the networks.

3.6.3 Automated Validation

The above offers a range of techniques for validating and therefore selecting a network for a learning task. In order to devise an automated mechanism for network selection, several issues must be taken into account. The fact that there are different methods available for validating a learning system suggests that the training procedure really acts as a means for generating a hypothesis and is therefore only one stage in the learning process. What has been shown in this chapter is that for a fixed neural network there is generally a variety of maps available for a given

learning task. It is therefore down to the validation procedure to select the best solution. If we adopt a strictly statistical approach to the validation procedure, and construct an automated system that selects networks on the basis of performance on a fixed out-of-sample validation set we run the risk of over-training. This was demonstrated in the examples given in 3.4, 3.6.1 and 3.6.1.1. Clearly, out-of-sample validation will play a major part in the eventual selection of a network. However, what we wish to avoid is the situation where the validation procedure dominates the design procedure to the extent that over-training is almost inevitable. One of the methods mentioned in 3.6.2 attempts to provide good network solutions through training rather than hypothesis testing. We now take a closer look at this method to see if it is possible to incorporate this method into an automated neural network design and selection system.

The method is the application of Occam's Razor. To apply Occam's Razor means that we accept that smaller nets generalise better. Ideally we would like some means to test this heuristic, particularly in light of the fact that MLPs have been shown to be universal approximators. If a large network contains the hypothesis space of a smaller network there is no reason to suppose that the larger network cannot match the smaller network's solution. Many researchers have suggested that network pruning has improved generalisation but to date no detailed set of experiments has confirmed this [MoUt92], [WeHR92]. One of the reasons for this is the fact that unless there exists a known neural network solution to a given problem, and that the solution is minimally descriptive, then it is hard to test the hypothesis that smaller networks generalise better than larger networks.

In Chapter 5 we come back to this issue and devise a means for testing Occam's Razor for neural networks. To do this we introduce a method for artificially generating test learning problems that have known neural network solutions using a genetic algorithm. Moreover, the known solutions are generated in such a way as to approximate a minimally descriptive network. This provides a means for testing different network architectures against the known solution in terms of their generalisation ability. Two results are achieved by these experiments: firstly we statistically validate the principle of Occam's Razor, and secondly we introduce and validate a method for automated network architecture design based on Occam's Razor (which we call Network Regression Pruning). These results are significant in terms of what has been discussed above. It means that the automated system is *not* totally reliant on the statistical evaluation of a neural network performance on an *out-of-sample* test set, and therefore insulates the process to some extent form over-training. As mentioned a genetic algorithm is used in the design of these experiments, and therefore we discuss this work in Chapter 5 after we have examined genetic algorithms [Chapter 4].

3.7 Summary

In this chapter we have taken a detailed look at the neural network training process. We started by listing the range of parameters associated with neural network training and then described the effects of each parameter on the neural network modelling process. By doing this we have seen that a number of parameters require careful consideration if: i) the trained network

is to achieve good generalisation, and ii) that validation results are to reflect accurately the performance of the trained network. Three main areas of concern have been raised. Firstly, an automated procedure of applying neural networks will require some method for determining data selection and data pre-treatment. Furthermore it was shown that data pre-treatment requires careful safeguards so as to avoid misleading results in the validation phase. Secondly, a systematic method for setting a network's architecture is required. It has been shown how a neural network can act as a memory look-up, and that from a theoretical viewpoint strong guidelines for designing a network architecture do not exist. Thirdly, it has been discussed that at present there are no strong guidelines for setting network training parameters (such as learning rates and number of training cycles), and that these too can produce misleading results. In summary, feed-forward neural networks have been shown to be sensitive to a variety of parameter choices.

In order to automate the application of a neural network to a given time series problem, we will require methods for systematically dealing with each of the above issues. In the next chapter we provide a detailed analysis of genetic algorithms, with the intention that genetic algorithms provide a general-purpose optimisation technique which may offer one way in which to tackle some of the neural network parameterisation problems.

Chapter 4

Genetic Algorithms

This chapter provides a detailed analysis of the Genetic Algorithm (GA) search technique. The relationship between the choice of GA operators and the likely success of a GA search is investigated. A new style of GA is introduced that makes use of multiple randomly selected representations during the course of a run. The Multi-Representation GAs use transmigration and transmutation operators to adjust the encoding of individuals within the population via base changes. It is shown that contrary to conventional GA analysis there is no formal justification for preferring binary representations to those of higher base alphabets. Moreover, higher base alphabets provide a simple method of dynamically re-mapping the search space. A series of experiments compares Multi-Representation GAs to the conventional binary Holland GA on a range of standard test functions.

4.1 Using Genetic Algorithms

There are two main reasons for wishing to include Genetic Algorithms (GAs) in an automated time series analysis system. The first reason relates to the potential GAs offer in terms of an adaptive control mechanism. Chapter 2 provided evidence for this quality, citing various examples of GAs with specific reference to financial applications. The second reason relates to the difficulty in optimising a Neural Network (NN) application. These problems were explored in Chapter 3 and relate to neural network parameterisation. The neural network parameterisation issue is now a well recognised problem [Chapter 3] and has led many researchers to advocate secondary optimisation techniques as a means for combatting the types of problem raised in Chapter 3 [GoKh95], [Davi91].

Before we can design a specific form of GA to be used in conjunction with a neural network for automated time series analysis, it is important to establish the broader design issues involved in using GAs. It is the purpose of this chapter to focus on the specifics of GA design so as to lay the foundations for the methods for combining neural networks and GAs that will be used in Chapter 5. To this end, this chapter is structured as follows: we start with a general description of the basic GA and review existing GA theory. We then describe some of the problems associated with the design, or paremeterisation, or a GA application. This includes a general discussion on "the shape of space", in terms of search algorithms and traversal operators. We point out that it is the combination of representation and traversal operators that define an algorithm's *view* of a given search problem, and hence *gives rise* to a fitness landscape. In this sense, all notions of search difficulty, such as modality (number of peaks in a landscape) are algorithm dependent, what one algorithm will find hard, another may find easy, and *vice versa* [HoGo94], [MaWh93], [RaSu95], [WoMc95], [KiDe95].

It is suggested that randomly remapping space via base changes provides a simple means of applying multiple search strategies to a given search problem, and that this offers a pragmatic means for probing a fitness function from many *views*. We introduce a number of new algorithms

based on two new operators, *transmutation* and *transmigration*. Both operators relate to randomly, and dynamically, adjusting the encoding of an individual during the course of a GA run. This gives rise to family of *multi-representation* GAs which re-map the search space via base changes as the search proceeds. This technique is demonstrated and compared to both random sampling and the standard Holland binary GA [Holl75], [Gold89], [Davi91], [Mich92], on a range of standard cost functions.

4.2 Search Algorithms

Typically a search problem is posed as an optimisation task in which a cost, or fitness, function must be maximised or minimised, subject to various parameter constraints. Recent interest in the development of general-purpose computational search techniques has broadened the scope of stochastic search algorithms. From *stochastic hill-climbers*, *tabu search*, *simulated annealing*, *evolutionary strategies*, and *GAs*, hybridisations have emerged, such as "recombinative simulated annealing" [MaGo92] and "dynamic hill climbing" [YuMa93], which blur the distinctions between the different approaches. When looked at in their underlying form, what all of these techniques share in common is a representation – a way of encoding candidate solutions to the problem – a problem-specific objective function for evaluating the "fitness" of candidate solutions, and traversal operators used to lead the search. Loosely speaking, traversal operators fall into four main categories:

- *Neighbourhood* operators aim at fine detail searching, with a localised view of the space.
- *Explorative* operators provide a broader attack, encouraging search in new areas.
- *Recombinative* operators combine material from points in the solution space to produce new candidate solutions.
- *Selection* strategies determine the acceptance or rejection of new points, based on their fitness. Selection operators determine (confine) the area of search and as such may also be viewed as traversal operators.

However, the concept of "closeness" or "neighbourhood" can be shown to be operator and representation dependent. It is this combination of representation and traversal operators that define an algorithm's *view* of a given search problem, and hence *gives rise* to a fitness landscape[1]. A search technique's representation and operators reflects the assumptions it makes about the search space, and hence determines a bias in the search trajectory. If this bias happens to align with the given fitness problem then the search will generally succeed; if not, the search can be misled. One way to tackle this problem is the dynamic use of multiple search heuristics. Whilst this strategy offers no shelter from the formal limitations on search algorithms [RaSu95], [WoMc95], it does provide a pragmatic means for probing the space of search operators that may be suited to a specific search problem. We demonstrate some of these issues for GAs. In

[1] A similar argument has been adopted by [Jone95].

particular, we show that GAs have attractive features in terms of the way a landscape can be adjusted, and that this can provide a simple means for applying multiple search strategies to a given search problem. The purpose of this investigation is to provide an analytic background so as to make an informed decision on how best to combine neural networks and GAs for automated time series modelling system. We start with a general description of the way in which GAs have been traditionally analysed in terms of the algorithms search strategy.

4.2.1 The GA Search Process: The Simple GA

As was the case for neural networks, the number of parameters associated with a GA application can be seen as an optimisation process in its own right [Mühl91]. The way operators are defined, the parameter settings they take, and the representation used for the individuals all contribute to the likely success of the algorithm. There has been considerable theoretical analysis of GAs over the last 20 years, and in this section we examine some of this analysis in an attempt to distil practical guidelines for applying GAs.

It is important to note that there is a huge variety of GA styles that are currently being used [DeKi93], and yet much of the analysis of GAs has tended to apply to a limited class of *simple* GA [Holl75], [Gold89], [Mich91]. The simple GA is characterised by three main attributes: firstly it manipulates fixed length binary strings which belong to a fixed population size. Secondly, the algorithm uses two-parent crossover which preserves bit positions along the length of the string. Thirdly, the algorithm employs a form of *roulette-wheel* selection with *generational replacement* [Gold89]. Roulette-wheel selection is a scheme by which members of the population are accredited with a probability of being selected in proportion to their relative fitness as compared to the whole pool. That is if $F=\sum_{i=1}^{N} f(x_i)$ is the total fitness for the whole population, then string x_i [Chapter 2] is assigned selection probability $f(x_i)/F$. Generational replacement refers to the selection regime in which each new generation completely replaces the existing generation, i.e., all members of the new generation have passed through selection and the biological operators (which may, or may not, leave them intact) [Chapter 2]. A final point which is also typical, is that the mutation rate is taken to be the probability that a bit position on any string is flipped between generations.

Each of these defining conditions affect the likely outcome of a particular GA run, and yet none of them are intrinsic to it actually working. For example, it has been demonstrated on many occasions that GAs with radically different characteristics than those of the simple GA are still effective optimisation techniques. Goldberg [GoDK91] has used *messy* GAs in which the string length is not fixed. Real-valued encodings are also commonly used [Gold89], [Davi91], [MuSV92a], [EsSc92] as are multiple pools with migration operators [MuSV92b], neighbourhood mating schemes (the diffusion model) [EaMa93], [MuSV92b], and many forms of selection [Whit89], [Davi91], [Sysw92]. Despite this diversity much of the analysis of the simple GA carries through to most of these models [LiVo92], [Radc92], [Gold89]. However, it is important to understand how each of these choices affect the GA search, and how each introduces

some form of bias into the search trajectory favoured by a given algorithm. We start with a restatement of standard simple GA analysis [Holl75].

4.2.2 Schema Analysis

For a simple GA using fixed-length binary strings the search space consists of 2^L possible points. The fact that all solutions reside on the 2^L hypercube means that there must be a transformation between the corners of the hypercube and the fitness domain. This implies that for a continuous fitness domain the search resolution is fixed by the length of the binary strings. The fact that a population of binary strings reside on a hyperplane of the 2^L hypercube is extremely fundamental to traditional GA analysis, so much so that a new symbol * is introduced to signify a similarity template for strings [Gold89]. The *match all* symbol represents either a 1, or a 0 in the bit string, and the result is referred to as a *schema* [Holl75], [Gold89]. Holland's analysis of GAs centres on the notion of competing schemata, or hyperplanes, and has resulted in the following formulation. Ignoring mutation and crossover and using roulette-wheel selection we have: for a population of size N the expected number of strings in generation g_{t+1} that reside on hyperplane H_k is given by:

$$E(H_k,t+1) = M(H_k,t).N.\frac{f(H_k,t)}{F}. \qquad 4.1$$

$E(H_k,t+1)$ is the expected number of strings that lie on hyperplane H_k at generation $t+1$, and $M(H_k,t)$ the number of strings of the population that lie on H_k at generation t. $f(H_k,t)$ is the *average* fitness of the population's strings on H_k at time t, and F is the *sum* of the fitness of the total population. That is, we simply calculate the average probability of being selected for members of a particular hyperplane, for a given generation, and then multiply it by the sampling number N. It means that if the hyperplane has above average fitness compared to the whole population, then the expected number of strings belonging to the hyperplane grows. If we assume (for the sake of demonstration) that hyperplane H_k is above average by a fraction of at least ε (that is $f(H_k,i) = F/N + \varepsilon.F/N$) for each of t generations, we should expect a growth in the members of H_k to be given by $E(H_k,t) = M(H_k,0)(1+\varepsilon)^t$, i.e., an exponential increase.

To include the effects of mutation and crossover into the hyperplane analysis the notion of a schema's *defining length* has been introduced [Gold89]. This is defined as the number of possible places along the length of a schema that may be disrupted by crossover. For example, the schema 110**** has defining length 2, where as 11*0*** has defining length 3. The probability of crossover disrupting a hyperplane is therefore given by $P_{dcross}(H) = P_c\, \partial l\, H)/(L-1)$ (where ∂l is the defining length of the schema/hyperplane and P_c is the crossover probability and L the length of the string). Another possibility is for the hyperplane to be disrupted via mutation. If we call the number of fixed positions on the hyperplane its *order* [Gold89] (i.e., this is the number of non-star variables) then the probability that mutation will *not* change one of these values is given by $P_{no\ mute}(H) = (1-\mu)^{ord(H)}$, where μ is the probability of mutation. For small values of μ we can approximate this by $1-ord(H)\mu$. From these expressions the standard expected hyperplane growth rate with small probabilities of mutation approximates to:

$$E(H_k,t+1) \geq M(H_k,t).N.\frac{f(H_k,t)}{F}\left(1-\frac{\partial l(H_k)}{L-1}.P_c - ord(H_k)\mu\right). \quad 4.2$$

The inequality is a result of ignoring higher order terms involving the probability of mutation. This equation is known as the *schema theorem* and it gives rise to the observation that above average schemata with short defining length and low order steer the GA search trajectory. This idea has been more formally dubbed the *building block hypothesis*, [Gold89], [Mich91]. Michalewicz describes this as, "A genetic algorithm seeks near-optimal performance through the juxtaposition of short, low-order, high performance schemata, called building blocks" [Mich91].

The building block hypothesis has led to many forms of GA analysis. The first of which was the notion of implicit parallelism. Implicit parallelism was first described by Holland [Holl75] and refers to the way in which a single bit string simultaneously samples 2^L schemata. It is argued that a single string is therefore a representative of many schemata, and that the search process, or specifically the process of constructing each new generation, takes into account multiple levels of competition. Goldberg [Gold89] has approximated the likely number of schemata that are *usefully* processed to be of order N^3 (taking into account the probability of disruptive effects from crossover and mutation). Disruption of building blocks has led many researchers to investigate the various effects of different forms of crossover. This has concentrated on ways of aiding the GA search in order to better form and retain building blocks throughout the search trajectory [SpJo91], [Spea92], [BrGo87], [Beth81]. The building block hypothesis has also given rise to the idea of *deceptive problems* [Gold89], [Beth81], [LeVo91], or problems that GA will find hard to solve. The defining characteristic of a deceptive problem is that hyperplanes guide the GA search to points that are not globally competitive. This has implications on the type of representation used by the GA (i.e., the encoding of the search space) and, as will be demonstrated later, highlights the fact that this is a parameter of the algorithm.

4.2.3 Building Blocks Under Review

There have been a number of recent criticisms of the schema theorem and the building block hypothesis, particularly when they are accompanied by claims of optimality. Grefenstette [Gref92] has warned against the use of a *static* version of the building block hypothesis in forming conclusions about GAs in general. He points out that the GA is a dynamic system, and that the population is only a sample of any particular hyperplane and is therefore subject to variance. Moreover, the allocation of search time given by the GA to particular hyperplanes in the cases of high variance can work against the so-called *optimal allocation of trials* (which other researchers have asserted is a consequence of the schema theorem). Further, Mühlenbein [Mühl91] has stated that the estimate of fitness of a schema is in general not its real fitness, and that therefore if the schema theorem is viewed with this in mind it is almost a tautological statement describing selection. Deceptive problems have also been criticised in this light, in that most are designed with a static building block approach and therefore in practise may, or may not, be GA-hard, and that only in small problems where the population size is significant in terms of the schema can the effect be truly demonstrated. This last point has been underlined by the

fact that if GA-hard problems can be defined, then so can GA-easy problems. However, GA easy problems have been shown to be problematic to good convergence [FoMi92], [Sysw92].

Despite these criticisms the schema theorem and building block hypothesis have led to a deeper understanding of the GA search method. Moreover, the schema theorem, even when viewed as a simple statement of sampling probabilities, implies that the GA representation in conjunction with the biological operators combines to make certain search trajectories more probable than others, and that this fact must be taken into account when designing a GA application. In the next section we demonstrate some of these issues and look for ways in which to experiment with different GA designs.

4.3 GA Parameters

In designing a GA application we are confronted with a range of parameter choices. To make effective choices we will require an understanding of the sensitivity of a GA to each of the possible parameters. An aspect that has appealed to GA users is the fact that a GA working on a population of bit strings has no knowledge of the semantics associated with these bits. Its only contact with the environment is through the global fitness measure associated with the entire string. This is not to say that GAs work equally well on all problems, only that these differences can be attributed to the underlying search space rather than the semantics of the problem domain [Beth81], [BeMS90]. However, what is less well documented is the role the GA operators play in defining the nature of a particular search space. Moreover, if by coincidence the structure imposed by the operators happens to align with some underlying features in the fitness landscape, then the algorithm performs well. Conversely, a mismatch between operators and fitness landscape can produce poor results. This fact has recently been shown to be true for all search methods [WoMc95]. This result goes as far as to show that all search techniques are equivalent in terms of finding a global optimum when viewed across the entire space of possible fitness functions. Once more this result serves to emphasise the importance of finding a search algorithm that *fits* a given search problem if good results are to be achieved. In the next few sections we shall give some examples of this effect for GAs. Once we have done this we will be in a position to propose a variation of the standard GA which attempts to counter some of the problems raised in GA parameterisation. As a first step towards this end we start by making explicit the role played by the representation that is manipulated by the GA.

4.3.1 The Shape of Space

We have mentioned the fact that the biological operators impose a shape on the *solution space* preferred by the GA. To make this idea concrete we take a binary bit string GA using a form of crossover that preserves bit positions. The most common forms of crossover that are included in this are one-point crossover (described in Chapter 2), uniform crossover (crosses the alleles between two parents independently at each locus with some probability), two-point crossover (selects two loci at random and crosses the string segments between these positions on both parents), N-point crossover (as for two-point except a number of sections are specified) [Gold89].

The following observations would also hold for more elaborate multi-parent crossover, provided bit positions are maintained.

If we start with a fixed population and only apply crossover then clearly we can only ever achieve a permutation of the bits that are present at each locus of the starting population. This has the effect of trapping the search on the smallest containing hyperplane. Each sub-cube, face, edge, etc., of the hypercube represents a boundary to the search available to simple crossover. That is to say, simple crossover can only ever explore the values of the smallest hyperplane containing the population. Once an encoding of the space is imposed, the effects of mutation, crossover, and the concept of 'nearness', are encoding dependent. For binary encoded decimal this effect is based on the Hamming distance, so that small changes in the encoded space can correspond to large changes in the decoded space. For example, the binary string 00001 represents the value 1, and the single mutation in the leading bit position gives 10001, which in Hamming terms is distance 1, but in the decoded space gives the value 17, a distance of 16. This re-defining of the shape of the search space through an encoding is extremely fundamental in terms of the GA search trajectory, not least because it dictates the way in which crossover and mutation move within the search space.

To illustrate this we take an example of a so-called GA deceptive problem. Before we give this example it should be noted that these effects are independent of the criticisms made above about the use of the static building block hypothesis. The fact that a GA is a stochastic search process means that deception can never be total, in the sense of guaranteed convergence to a sub-optimal points. However even with the sampling effect created by the use of a population, and the fact that a variance will be associated with each hyperplane sampled, certain probabilistic effects can always be manufactured so as to probably mislead the GA search.

Table 4.3.1 gives an example of a deceptive problem. The reason that it is considered deceptive is that if the utility (average fitness) [Gold89] of the hyperplanes containing the optimum (at 111) are compared to the utility of the hyperplanes containing the deceptive point optimum (at 000) then in all cases the deceptive hyperplanes have higher utility. This in turn implies that the GA search has an increased probability of being directed away from the true optimum and towards the point 000.

Encoding	Base 10	Fitness	Hyperplane	Members	Utility	Hyperplane	Members	Utility
000	0	6	00*	0,1	4 5	11*	6,7	4
001	1	3	0*0	0,2	4	1*1	5,7	3.5
010	2	2	*00	0,4	4	*11	3,7	3.5
011	3	0	**0	0,2,4 6	2 75	**1	1,3,5,7	2.5
100	4	2	*0*	0,1,4,5	3	*1*	2,3,6,7	2.5
101	5	0	0**	0,1,2,3	2 75	1**	4,5,6,7	2.25
110	6	1						
111	7	7						

Table 4.3.1: The function DF1 using standard binary encoding.

The way in which crossover, mutation and the representation enforce the characteristics of the search space can be seen by two simple demonstrations. Firstly, it has been pointed out [LiVo91] that a re-ordering of the encoded space can transform the relative difficulty of the problem. This

has the effect of permuting the points on the hypercube. So for example if a new binary code is used for DF1 we can place optimal points on a single hyperplane, which in turn implies that such sets become stable under crossover. Table 4.3.2 gives an example of the transformed function DF1. Secondly, a similar effect can be achieved by changing the base of the representation. So again using DF1, a switch to a standard base 3 encoding transforms the space to a base-3 hypercube. This is depicted in Figure 4.3.1 alongside the usual binary cube. The base 3 encoding has new hyperplanes, and therefore new stabilities under simple crossover.

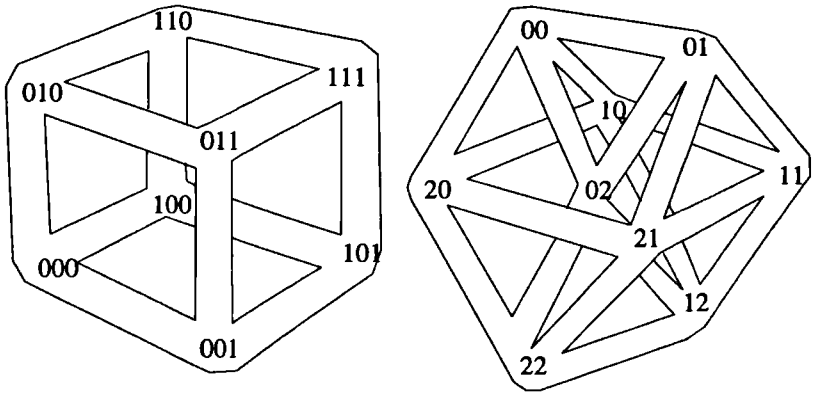


Figure 4.3.1: Binary cube and Base 3 hypercube.

To describe regions that are crossover invariant in the base 3 hypercube requires an extension the usual * notation. We let $*_1 = 0$ or 1, $*_2 = 0$ or 2, $*_3 = 1$ or 2, and $* = 0,1$ or 2.

Encoding	Base 10	Fitness	Hyperplanes	Members	Utility	Hyperplane	Members	Utility
000	0	6	00*	0,7	6.5	11*	3,5	0
001	7	7	0*0	0,2	4.0	0*1	1,7	5
010	2	2	*00	0,6	3.5	*01	4,7	4.5
011	1	3	0**	0,1,2,7	4.5	1**	3,4,5,6	0.75
111	5	0	*0*	0,3,6,7	4.0	*1*	1,2,3,5	1.25
110	3	0	**0	0,1,2,3	2.25	**1	1,4,5,7	3
101	4	2						
100	6	1						

Table 4.3.2: Permuted binary encoding for the function DF1.

Encoding	Base 10	Fitness	Hyperplane	Members	Utility	Hyperplane	Members	Utility
00	0	6	$0*_1$	0,1	4.5	$*_1*_1$	0,1,3,4	2.75
01	1	3	$0*_2$	0,2	4	$*_1*_2$	0,2,3,5	2
02	2	2	$0*_3$	1,2	2.5	$*_1*_3$	1,2,4,5	1.75
10	3	0	$0*$	0,1,2	3.66	$*_1*$	0,1,2,3,4,5	2.16
11	4	2	$1*_1$	3,4	1	$*_2*_1$	0,1,6,7	4.25
12	5	0	$1*_2$	3,5	0	$*_2*_2$	0,2,6,8	2.25
20	6	1	$1*_3$	4,5	1	$*_2*_3$	1,2,6,7	3.25
21	7	7	$1*$	3,4,5	0.66	$*_2*$	0,1,2,6,7,8	3.16
22	8	0	$2*_1$	6,7	4	$*_3*_1$	3,4,6,7	2.5
			$2*_2$	6,8	0.5	$*_3*_2$	3,5,6,8	0.25
			$2*_3$	7,8	3.5	$*_3*_3$	4,5,7,8	2.25
			$2*$	6,7,8	2.66	$*_3*$	3,4,5,6,7,8	1.66

Table 4.3.3: Base 3 representation of DF1: showing some of the hyperplane competition.

To see the effect of a base change on the function DF1, Table 4.3.3 gives the usual base 3 encoding along with the fitness values associated with each point, and the utility of some of the new hyperplanes. Note that in both the permuted binary coding, and the base 3 representation, the utility of hyperplanes containing the true optimum (fitness 7) are increased relative to deceptive hyperplanes (containing point 0, fitness 6). The effect of this can be seen in simulation. Figure 4.3.2 gives the convergence profile for all possible 3-point starting populations for each of the representations. As can be seen the usual binary encoded only converges to the optimal point in 23% of the time as compared to 53% and 63% for the base 3, and the re-ordered case respectively.

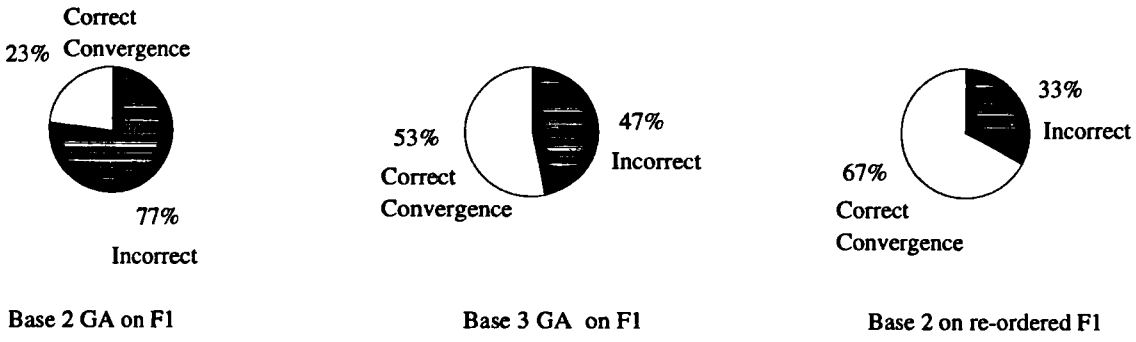


Figure 4.3.2: DF1 convergence using 3-types of GA encoding.

There are several issues raised by the above. Firstly, it should be pointed out that to make the optimal transformation of the binary code is impractical for real-world problems. There are $2^L!$ possible codings (where L is the string length) of the binary cube. This is clearly far larger than the search space itself, and unless detailed knowledge of the problem is given such a transformation is impractical. Secondly the transformation to base 3 encoding will not always result in a smoother passage for convergence, this too is dependent on the actual problem. An example of this given by the test function DF2 described in Syswerda [Sysw92], [Whit91]. It consists of 10 deceptive 4-bit problems concatenated and summed together (see Table 4.3.4). The deceptive peak for each for the sub-problems is at 28 for bit pattern 0000, providing a maximum deceptive value of 280 for the 40 bit length string. The true maximum has a value of 30 for bit pattern 1111, providing a global maximum of 300.

Bit value	Fitness	Bit value	Fitness	Bit value	Fitness	Bit value	Fitness
1111	30	0100	22	0110	14	1110	6
0000	28	1000	20	1001	12	1101	4
0001	26	0011	18	1010	10	1011	2
0010	24	0101	16	1100	8	0111	0

Table 4.3.4: The function DF2. The above gives the fitness of each of 10 4-bit deceptive problems.

To create a stability for DF2, unlike DF1, we require a switch to base-5 which means the two highest peaks for each of the sub-problems (at points 0 and 15 in base-10) are represented in base-5 by 00, and 30 receptively. By switching to base-5 we create a stable edge (under crossover) on the base-5 hypercube. Figure 4.3.3 compares the convergence profile for the binary and base-5 representations. As can be seen the base 2 representation fails to converge to the correct optimum, whereas the base-5 representation succeeds.

Syswerda [Sysw92], has studied DF2 using different styles of GA, testing different forms of crossover (i.e., one-point, two-point, uniform etc.), and population sizes. He compared the convergence profiles of all algorithms along with a control random search. In all cases he used the standard binary encoding, and in all cases each GA failed to converge to the correct optimum (the experiments where run to 5000 generations). Not only does this highlight the importance of the *correct* representation, it also makes clear the fact that deception is algorithm specific. That is to say, no single representation is optimal for all problem instances, and that for real-world problems the representation chosen is crucially linked to the likelihood of the algorithm finding good (optimal) solutions. This observation however, does not give any clue as to how best to go about finding good representations. This is particularly important when little is known *a priori* about the target function, as will be the case in real-world problems. However, one possibility that is raised by this observation is the use of multi-representations within the course of the GA run. That is, we use many different representations as a means of probing the fitness function in an attempt to find a representation that is suited to the given problem.

Dynamic remapping of the search space has been applied before [MaWh93], [CaSE89], [ScBe90], [Shae87], however in all such instances the algorithms have used forms of permuted binary encoding, and have used a directed switching mechanism between representations. The justification for binary codes has been based on the idea that binary codes increase the hyperplane sampling frequencies during the search [Holl75], and are therefore optimal [Holl75].

In the next section we show that this is untrue if all hyperplanes created by a switch to a higher alphabet are counted [Anto89]. Moreover, we suggest that higher base alphabets are a simple way of remapping space, and that they dispense with the need for some of the complex mapping strategies required in order to switch between permutations of a binary code.

Later in this chapter we shall formulate and test a GA search strategy that makes use of multiple representations via base changes. Before we do this we first examine some of the issues surrounding the use of higher base alphabets, and review existing GA operators.

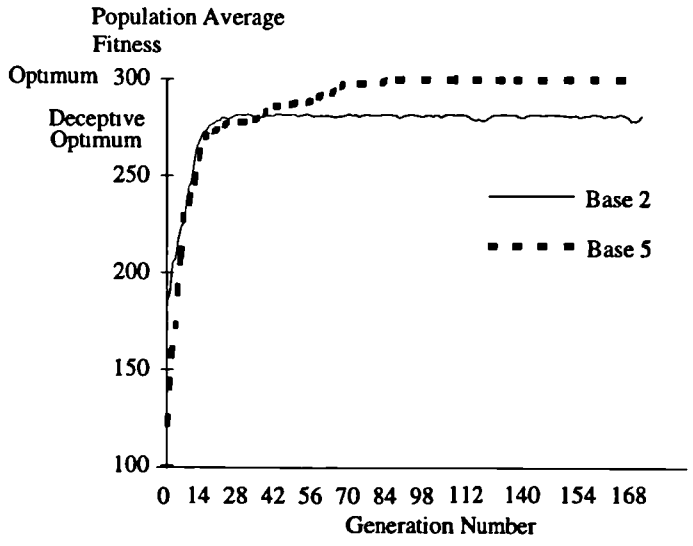


Figure 4.3.3: Base 2 and Base 5 encodings on DF2. Population size of 20.

4.3.2 Population Encodings

As mentioned above, one of the reasons that binary alphabets have been traditionally favoured by the GA community relates to the idea that the smallest possible alphabet creates the largest possible hyperplane sampling frequency [Holl75]. However, Holland's original derivation for the binary alphabet ([Holl75] pp. 71 para 2.) did not include all of the hyperplanes created once a higher base alphabet is used for encoding (independently derived using a different method in [Anto89]). This relates to the inclusion of new * star variables that are required in order to partition the search space into the full set of hyperplanes and hypercubes. Interestingly the sampling frequency increases for larger alphabets if the full number of hyperplanes, or schemata, are taken into consideration. For example, if we take two alphabets with $a_2 > a_1$ and $a_1 \geq 2$, then the full number of hyperplanes containing at least two points (and therefore requires a * variable) is given by;

$$\sum_{i=2}^{a_1}\binom{a_1}{i} = 2^{a_1} - a_1 - 1 \text{ and } \sum_{i=2}^{a_2}\binom{a_2}{i} = 2^{a_2} - a_2 - 1, \tag{4.3}$$

respectively. This implies that the full number of hyperplanes for each hypercube of dimension L (and therefore strings of length L) will be given by;

$$(2^{a_1} - 1)^L \text{ and } (2^{a_2} - 1)^L. \tag{4.4}$$

In order to compare the size of these terms with fixed precision over the search space let $L = L_1$ for a_1. To achieve the same precision for a_2 we then require a string length of,

$$L_2 = \log_{a_2}(a_1^{L_1}), \text{ which gives } a_2^{\log_{a_2}(a_1^{L_1})} - 1 = a_1^{L_1} - 1, \tag{4.5}$$

and therefore the same highest decoded value in both encodings. For the two encodings we therefore have the following number of hypercubes, hyperplanes, and points, $(2^{a_1} - 1)^{L_1}$ and $(2^{a_2} - 1)^{L_2}$ respectively. From this it is straightforward to derive that,

$$(2^{a_2} - 1)^{L_2} \geq (2^{a_1} - 1)^{L_1} \tag{4.6}$$

To see this, without loss of generality let $a_2 = a_1^k$, for $k \geq 1$ and $a_1 \geq 2$. Using the identity given for L_2 in equation 4.5 and noting that $\log_{a_2}(a_1^{L_1}) = \dfrac{\log_{a_1}(a_1^{L_1})}{\log_{a_1}(a_2)}$, we can take logs and re-write 4.6 as follows:

$$\log_{a_1}\left[2^{a_2}(1 - \frac{1}{2^{a_2}})\right] \geq \log_{a_1}(a_2).\log\left[2^{a_1}(1 - \frac{1}{2^{a_1}})\right], \tag{4.7}$$

which re-arranges to the following;

$$a_2 \log_{a_1}(2) - a_1 \log(a_2) \log_{a_1}(2) \geq \log_{a_1}(a_2).\log\left[1 - \frac{1}{2^{a_1}}\right] - \log_{a_1}\left[1 - \frac{1}{2^{a_2}}\right], \tag{4.8}$$

which using the identity for a_2 simplifies to,

$$\frac{a_1^k}{k} - a_1 \geq \frac{\log_{a_1}\left[1-\frac{1}{2^{a_1}}\right]}{\log_{a_1}(2)} - \frac{\log_a\left[1-\frac{1}{2^{a_1^k}}\right]}{k\log_{a_1}(2)}. \qquad 4.9$$

From 4.9 we can see that if $k=1$ the equality holds (i.e., both alphabets or encodings are the same), and that for $k>1$ we note that the left hand side of 4 9 is negative since $1-\frac{1}{2^{a_1}} < 1-\frac{1}{2^{a_1^k}}$. Conversely the right hand side of 4.8 is always positive (recall that $k-1 \geq \log_{a_1}(k)$ [Binm77]).

We therefore have a larger hyperplane sampling rate for larger alphabets. This result is in direct contrast with earlier GA analysis [Holl75], [Gold89]. Many researchers have suggested that binary alphabets are best in that they ensure the largest number of hyperplanes sampled by any individual of the population. What 4.9 shows is that this is not true if all hyperplanes are counted, which in turn means all * variables must be included. In Holland's original analysis [Holl75], and in much of the research that has followed, only a single * variable has been taken into account when discussing higher base alphabets. This however, does not reflect the full number of similarity templates that exist. A visual interpretation of this is given in Figure 4.3.1. The base 3 cube is not simply the base 2 cube with an extra point appended, there are new stabilities under crossover in that a face of this cube consists of three points as opposed to 4 points in the base 2 cube. In this sense, once a full hyperplane analysis is taken into account traditional GA analysis should advocate large as opposed to small alphabets. However, as has been mentioned a GA is dynamic system and therefore hyperplane sampling rates and the utility values of hyperplanes are not always useful in predicting the behaviour of the algorithm [Gref92]. Therefore how much benefit we gain from this analysis ultimately rests on the sampling characteristics associated with each hyperplane and the particular fitness function under consideration. That is to say, on real-world problems the sampling variance of each of the hyperplanes will have far more influence on the search trajectory, than simply counting the number of hyperplanes that may be sampled. Again it leaves the best strategy for real-world problems very open, since the variance of fitness over hyperplanes will by definition be unknown. What does emerge from the above is the importance of representation, in conjunction with mutation and crossover. If a *good* representation is chosen that coincides with the problem instance then the algorithm efficiency in terms of the search is improved dramatically (for example, in base-5 DF2 is solved in under a 200 generations, whereas using base 2 it is not solved by generation 5000 [Sysw92]).

What the above suggests is that there can be an advantage in using higher base alphabets, and that in some instances a change of base will aid the of the search process (i.e., DF1 and DF2). Before we describe a method for exploiting this possibility we first provide some background to other parameter choices within the GA, namely the population size, crossover, mutation, and selection regimes.

4.3.3 Crossover, Selection, Mutation and Populations

The parameter choices for a GA outside that of the representation, are the crossover method, the selection scheme, the mutation style, the population size and the number of generations or halting criteria. There has been work in all of these areas, but hard rules for optimal settings do not exist, which again relates to the difficulty in defining generic rules for all problem instances. Some of the work in this area is summarised below.

One of the features of the Holland GA is the fact that crossover preserves loci at allele positions. In terms of building block formation this is has been regarded as an important property [Gold89], in that allele positions that have converged can only be disrupted via mutation. However, a number of researchers have suggested that crossover invariant subsets can be created through many different, and possibly complex, definitions of the operator [Radc92], [DeKi94]. This type of analysis has also been used to argue for more flexible representations that reflect more precisely the type of search space that is being traversed [Davi91]. By taking representations that are more intuitively linked to the problem space it is easier to design operators incorporating problem specific knowledge. This last point has been made by many researchers [Radc92], [Mich92], [Davi91], [WoMc95] particularly in respect to real-world problems.

One effect that has been studied fairly extensively is the choice of selection regime [Gold89], [Davi89], [Mich91]. A problem that occurs with the roulette wheel method relates to the range of values taken by the fitness function, and the manner in which selection pressure varies over the course of a run. Whitly [Whit89] has argued that there are two primary factors in genetic search, the population diversity and selective pressure. He points out that many of the parameters that are used to *tune* the GA search are ultimately means for adjusting the selective pressure, and diversity. As selective pressure is increased the search becomes confined, and diversity is lost. This is seen as *exploitative* in that the algorithm is responding in a very immediate way to the information provided by the population. Equally, reducing selective pressure increases diversity, and this is seen to correspond to the algorithm becoming more *explorative*, in that its search is less affected by fitness evaluation. The balance between these two extremes is a subject area in its own right [Mich91], and there are many techniques that have been developed which have certain advantages in certain circumstances. To summarise these effects Table 4.3.5 gives an example population for a fitness function, F1.

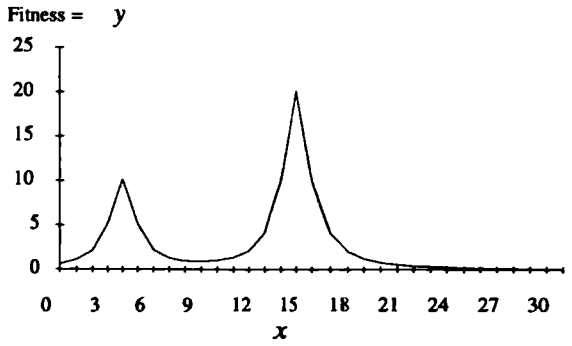


Figure 4.3.4: Simple fitness function F1.

F1 is given by the equation;

$$y = \frac{10}{(4-x)^2+1} + \frac{20}{(15-x)^2+1}. \qquad 4.10$$

F1 has peaks at $x = 4$ and $x = 15$, with the latter the global optimum. The way in which the example population effects the GA search is dependent on the selection regime that is employed. A sample of common selection methods are given below with the effect of each on the example population for F1 given in Table 4.3.5.

Decoded Population Member:Generation *G*1	Binary	F1: Fitness
0	00000	0.67
1	00001	1.1
2	00010	5.13
9	01001	0.925
12	01100	2.15
19	10011	1.22
25	11001	0.22

Table 4.3.5: Example population for function F2.

Roulette: Sum the total fitness of the whole population, call this *S*. Then the probability of selection for a member with fitness *f* is given by *f*/*S*.

Truncated Roulette: As for roulette except either i) restrict population to some fraction of the best individuals in the current generation (this is the style shown in Table 4.3.6 with roulette selection only taking place between the top 70% of the population), or, ii) copy a fraction of the best individuals into the next generation and select the remainder using roulette over all members of the current generation.

Window: Evaluate fitness of population members. Find the minimum fitness, assign each member a fitness according to how much it exceeds this minimum value. Calculate selection probability according to this new fitness evaluation.

Elitism: Copy best individual into next generation. Then use any other form of probability assignment for remainder of new population. In Table 4.3.6 this is done for roulette.

Linear Fitness: Order the members by decreasing evaluation. Create fitnesses that begin with a constant term and decrease linearly. The constant and the gradient of the line are parameters of the technique. Once a fitness has been assigned, calculate probability of selection via roulette. In Table 4.3.6 we use constant 2 and gradient 2/(population size).

Remainder Stochastic: Allocate space for individuals in the next generation according to the integer expectation of occurrence (according to roulette). For the remaining places the whole population competes as with usual roulette with the fractional parts of the expected values used to calculate their respective probabilities.

Tournament Selection: (Not depicted in 4.3.6) Choose some number of individuals randomly from the population (with or without replacement), select the best individual from this group. Repeat as often as desired. Tournaments are often held between pairs of individuals, although larger groups can be used.

Table 4.3.6 below gives the probability of selection for each of the population members given in Table 4.3.5. As can be seen from Table 4.3.6 (note: only the probability of being in the next generation for each member is given) the choice of selection method has dramatic effects on the likely search trajectory, at least in the short term. The long term effects, in the sense of how robust all of the above methods are in finding optimal solutions is still very much open, and the guidelines that do exist tend to rely on empirical study. The difficulty with empirical study in this instance is that by definition it can only encompass a limited set of fitness landscapes, and therefore suffers from the subjective choice of test functions. The example in 4.3.6 highlights the difficulty: if the selection pressure is increased the above may well converge to the sub-optimal peak as this is best represented in this population.

		Selection Regime: Probability of being selected					
Member	Fitness	Roulette	Truncated (0.7)	Window	Elitism	Linear Fitness	Remainder Stochastic
0	0.67	0.06	0	0.04	0.06	0.04	0.14
1	1.1	0.1	0.1	0.08	0.1	0.14	0.23
2	5.13	0.45	0.49	0.49	1	0.29	1
9	0.925	0.08	0.09	0.07	0.08	0.09	0.18
12	2.15	0.18	0.2	0.19	0.18	0.24	1
19	1.22	0.11	0.12	0.1	0.11	0.19	0.25
25	0.22	0.02	0	0	0.02	0	0.04

Table 4.3.6: The sampling probabilities for selection schemes for the population given in Table 4.3.5.

For elitism and stochastic remainder, the population member with value 2 has an assured place in the next generation. This means that a small population can quickly be dominated by this member. Alternatively, a decrease in selection pressure (represented by the roulette, truncated, linear and window schemes) could lead to *genetic drift* where the search takes little guidance from information received via the population, and the search becomes more akin to a random walk. Goldberg and Deb [GoDe91] have broached this issue analytically by comparing four selection methods in terms of their bias towards good solutions, growth ratio estimates (of good individuals), and take-over time computations (the time it takes for the population to be dominated by one individual, or one class of individuals that share the same fitness). As Goldberg and Deb point out, the analysis is largely descriptive and the question of which procedure is best is problem specific.

If selection schemes have proved problematic, then the same is true of population sizes and mutation rates. In these last two instances some progress has been reported, but mostly this has relied on some defining assumptions about the fitness landscape. This is not quite true of population sizes, where an estimate has been derived using basic sampling theory applied to building block, or schema analysis [GoDC92]. The idea is to use the Central Limit Theorem as a means for sizing populations on the basis of the sampling characteristics that are required in order to estimate the fitness of each building block. That is, populations are sized in order to minimise the variance associated with a finite (and incomplete) sample of a schemata's fitness so as to provide a good estimate of its mean. However, this does assume that the problem is non-deceptive, in that the mean fitness of a schema may not reflect the position of the true optimum.

Also a further difficulty is that for reasonable probability bounds on the likely error, large population sizes can result, which is prohibitive for complex fitness functions.

Mutation styles are variously defined [MuSV92a], [Gold89], [Mich92] but in principle it is a form of random change in a string's value. For mutation two main strands of research have been conducted. The first compares mutation with crossover, and compares the relative merits of both operators [Spea92], [FoAt90], [ScEs91]. From this, a mixed view emerges as to the relative advantage of one over the other. There is some consensus however, that mutation is vital, whereas crossover may have some advantage in some circumstances [Spea92]. One other aspect that is important in comparison of the two operators is the fact that crossover with selection leads to absolute convergence, whereas mutation on the whole will not. In the presence of mutation alone a cooling schedule (in which the mutation rate is dynamically adjusted to zero) would be required to guarantee convergence [FoOW66], [RiCo86], [BeMS90]. The second major strand of research is more practical in the sense that it relates to setting the actual mutation rate for a specific mutation scheme [Back93], [Mühl92]. Under certain assumptions it is shown that a mutation rate of $1/L$ (L the length of the string) is near optimal for unimodal functions. The derivation is based on a single fitness function (counting ones) for a binary representation. It looks at the probability of changing a string's wrong bits in order to create the optimum. However, again the difficulty involved in deriving mutation rates for multimodal functions has not been attempted, and Bäck [Back93] suggests the unimodal rate may not be appropriate.

In all of the above areas research continues, and clearly the subject of optimal parameter settings, and optimal operator choices is a subject area in its own right. With regard to this thesis one observation can be made that captures some of the issues: the difficulty in advocating any generic style of optimisation is that all optimisation techniques make some form of assumption about the solution they are trying to find.

In a practical sense, faced with a real-world problem, the time spent on searching for a good solution will ultimately be a function of the resource available, i.e., how much time is available to carry out the search, and what computational resource is at hand. At the end of that time the best solution found will the solution advocated. In a pragmatic sense, the best strategy for using the search time is to make multiple runs of different styles of algorithm in the hope that either they all agree on the optimal solution, or that the different assumptions made by the various algorithms pay off. In the two examples given above, DF1 and DF2, it was shown that the choice of encoding makes a drastic change to the shape of the search space. From this point of view it means that GAs provide an extremely flexible way of changing the assumptions the algorithm is making about the search space. It therefore provides a practical way of probing the problem from multiple directions. In the next section we devise a GA search strategy based on this approach.

4.4 A Strategy for GA Search: Transmutation

What section 4.3 has attempted to highlight is the practical difficulty involved in choosing, or designing, a generic search method that will perform well over a range of search problems. What transpires is the fact that the relative difficulty of a fitness landscape (as defined by any given

search algorithm) is algorithm dependent, and that as such, different search algorithms are suited to different search problems. This fact has been underlined by recent formal statements regarding the limitations of generic search techniques [WoMc95], [RaSu95]. It has been shown that in terms of finding a global optimum any two search algorithms can be shown to be equivalent over all possible search problems [WoMc95]. The validity of this result is easily noted if we allow arbitrary representations of the search space. In this case, it is always possible to permute the encoding of the search space retrospectively so that a given search trajectory in encoding space is forced to lead to the global optimum (i.e., create an encoding defined by permuting the algorithm's original solution point with the global optimum). This is more than a tautology—it says that for a given cost function a search heuristic can always be vindicated by the *correct* choice of encoding. Moreover, it says that an algorithm can never know when it is making a *correct* move, in that no search trajectory can be said to be better than another, since for any search trajectory there exists an encoding that leads it to the global optimum [RaSu95].

The practical implications of this for GAs is that deriving optimal parameters for each of the operators, or even deciding the actual choice of operators, is going to be problematic if little is known *a priori* about a search problem.

However, what has also emerged from the above is the relative ease by which a GA's method of attack for a given search problem can be re-shaped by changing operators or encodings. In a pragmatic sense this provides a simple way for creating and testing radically different search algorithms, that view, or define, the function landscape in new ways. If, by chance, the algorithm's view can be made to align with the search problem then algorithm will perform well. For real-world problems in which we know little about a given search problem, multiple search heuristics have a pragmatic quality. Only by testing the search problem from a number of views will it be possible to gauge the relative success of results. For GAs one way in which to do this is to dynamically re-map the space by using multiple encodings.

Remapping space has been discussed before [CaSE89], [MaWh93], [ScBe90], [Shae87]. One problem raised by remapping [LeVo90] is the fact that for a binary n cube there are $2^n!$ possible encodings. This makes the problem of finding a good encoding many times larger than the original problem. Two facts make this consideration largely redundant. In a formal sense, an algorithm that uses dynamic encodings is no worse than any other due to the formal restrictions placed on all search algorithms [WoMc95], [RaSu95]. In a more practical sense, perfect encodings are not necessary. If an algorithm's current point is amenable to search, then it does not matter what is happening in the remainder of the space. If the search is halted the encoding can be changed in an effort to find a bridge to a new area amenable to search regardless of the rest of the space. What is required from a remapping of space is the ability to free a given search from a local optimum so that the search can progress.

The existing methods for remapping the search space [CaSE89], [MaWh93], [ScBe90], [Shae87] have for the most part relied on some form of deterministic method for deciding, when and how remapping should occur. Moreover, for GAs, this has concentrated on ways of switching between different binary encodings of the search space. There are two main problems

encountered with this approach. Firstly, what grounds should be used for deciding what the new encoding should be? And secondly, how can arbitrary binary encodings be achieved? For the first point we have already discussed that there can be no formal justification for preferring one encoding from another for a given unknown search problem. This would suggest that a deterministic choice of encoding may be too rigid for arbitrary search problems. On the second issue, permuting binary codings also seems too restrictive, as it implies that an algorithmic scheme must be available for switching between arbitrary binary encodings. This in practice can prove problematic.

In contrast to these earlier schemes we introduce two variations: firstly we employ random changes in the algorithm's representation, and secondly we use base changes to remap the space. Random changes are advocated on the grounds that once a search heuristic has halted there is no basis by which to infer what the new representation should be (i.e., the algorithm may have found the global optimum). Base changes are advocated on the grounds that they provide a simple and practical way of remapping the space, so as to re-adjust sampling characteristics of each of the hyperplanes (as shown in section 4.3.2), and to open up new crossover invariant subsets under crossover, and open new lines of connection for mutation (as shown in section 4.3.1).

In terms of designing multi-representation GAs (MR-GAs) in which encodings are randomly generated, there are several possibilities, with two mains extremes. The first extreme would be to have all members of the population encoded using a different representation, and the other extreme would be that all members of the population use the same encoding, but that the encoding periodically changes. In all variations we wish to introduce the possibility of an individual (or a population) changing representation. To do this we introduce a *transmutation* operator. Transmutation acts in a similar manner to mutation, but instead of changing a string's, or individual's, encoded value by some random alteration, it changes its representation. For example the standard binary encoded value 1001, representing the base 10 value 9, could be transmuted to the encoding 21, again representing the base 10 value 9, except for the fact that it is now using base 4, the actual choice of new representation being randomly selected. The transmutation operator therefore, does not sample a new point within the search space it simply changes the way in which encodings of the search space are represented.

A third alternative as to a possible design of a MR-GA is to use a form of Parallel GA (PGAs) [Sten91], [MuSV92b], [CHMR87], [Norm88], [RaSu94] with different populations having different representations. The PGA is a variation of the standard GA, and makes use of multiple populations that communicate via a migration operator [MuSV92b], [CHMR87], [Norm88]. PGAs introduce the notion of space into the usual GA model. An extra parameter is appended to the individual placing it in space (i.e., each member of the population has a co-ordinate). Interaction between members is then restricted to neighbours having similar co-ordinates. Two main models exist: the first is the *Island Model*. Here several populations are allowed to develop in parallel occasionally exchanging members via migration [Sten91]. The second form of PGA is the *Fine-Grained Model*. Here every individual is given a unique co-ordinate representing space. Typically members are placed on a grid of some dimensionality with either a fixed or cyclic

boundary (e.g., a cube, globe, a lattice or torus) [EaMa93], [MuSB91], [GoWh93], [RaSu94]. Individuals only mate within a neighbourhood, or *deme*, and each deme overlaps with other demes depending on their size and shape. Characteristically, in-breeding within demes causes natural *niching* to occur [David91], which eventually break down as one neighbourhood emerges as the fittest and spreads throughout the population. This pattern of development is responsible for the models commonly referred to as diffusion models.

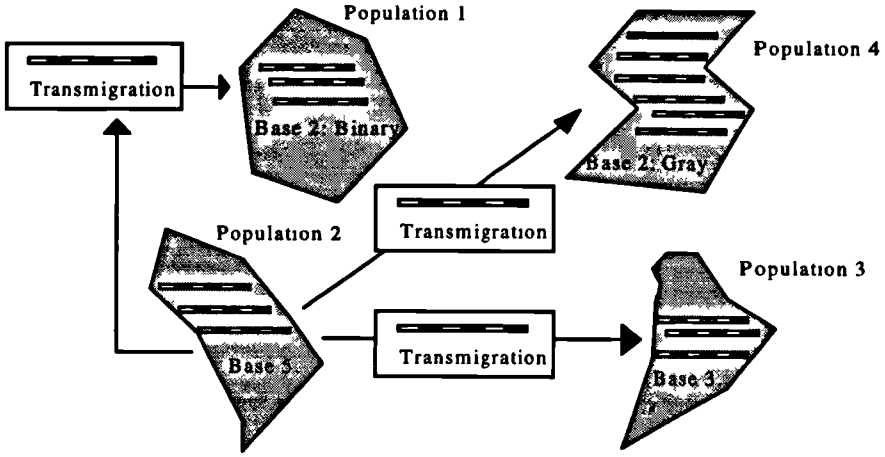


Figure 4.4.1: Multiple representations with communication via migration.

The PGA template offers a number of ways in which multiple representations can be used. The simplest variation is to attach a representation to the spatial co-ordinate. Here an individual's encoding is related to its spatial reference. To complete the algorithm's definition we introduce a *transmigration* operator which serves to move individuals between populations of different representations (Figure 4.4.1). Transmigration reduces to transmutation if no spatial reference is included.

In the next section we define and test a range of MR-GAs on suite of standard test functions.

4.4.1 Five New Algorithms: Multi-Representation GAs (MR-GAs)

To judge the various strategies that are available once a multiple representation strategy has been adopted, five MR-GAs were tested on 7 optimisation problems. The functions were chosen from what has become a common suite of GA test problems [Sysw93], [DeJo75], [Gold89], [MuSB91], [FoMi92], [KiDe95]. In each instance a fixed population, size 24, and fixed mutation rates, $1/L$ (where L is the length of the binary encoded string)[2] were used. All algorithms were subjected to 30 runs with different starting populations, and were run for the same number of fitness evaluations (10,000). All algorithms, unless otherwise stated, used standard 1-point crossover (with probability 0.6 [DeJo75]), and truncated selection (method 2 section 4.3.3, with fraction = 0.3). The MR-GAs are defined as follows:

MR-GA 1: Dynamic Representation GA (DRGA): This MR-GA uses a single pool (population - size 24). Each individual within the pool is encoded using an alphabet chosen at random from

[2]Note: the mutation rate was held at this value for representations used within the MR-GAs

the range 2 to 8. One-point crossover takes place in the representation of the individual with the highest fitness. The algorithm uses transmutation to adjust a members representation (each individual was transmuted with a probability 0.1).

MR-GA 2: Diffusion Model GA: One structured pool with diffusion across a ring. Individuals start in groups of base 2, 3 and 5 around the ring. Mates are selected for crossover with a likelihood inversely proportional to the square of the distance away from the individual. Crossover takes place in the representation of the individual with the highest fitness. Uses a transmutation (probability 0.1) operator in conjunction with normal mutation.

MR-GA 3: Statically-Sized Island Model GA (Island1): This algorithm uses three pools (size 8), each with a different representation (bases 2, 3 and 5) but constant across each pool. Transmigration every generation keeps pool sizes fixed swapping the overall best individual from one pool to the other two. The algorithm uses standard crossover within pools.

MR-GA 4: (MR-GA) Dynamically-Sized Island Model GA (Island2): Three pools (size 8), different representations (bases 2-3 and 5) but constant across each pool. Transmigration takes place between pools, with a probability 0.1. Pools sizes vary according to migration patterns.

MR-GA 5: (MR-GA) Seasonal Model GA: One pool, cyclic change of representation (randomly selected between base 2 to 8) every 30 generations.

One aspect that should also be mentioned with regard to the use of higher base alphabets is the possibility that a move to a new base over-specifies the space as compared to the binary representation. This technical issue was dealt with in two ways; firstly, new points sampled that were not defined within the original binary encoding were set to fitness zero. Secondly, large over-specification was prevented by restricting the range of the most significant decimal place within the new encoding. For example, the binary search space given by 11, 10, 01 and 00 (base 10 values 3, 2, 1 and 0 respectively) would translate to the base 3 codes 10, 02, 01, and 00 respectively. The string codes in base 3 given by 11, 12, would be set to fitness zero, and the codes 20, 21, and 22 would be prevented by restricting the leading place to values 0 and 1. This prevents some of the excesses of over-specification of the space. However, as will be shown despite this aspect the MR-GAs did not suffer in terms of their performance.

To test each of the algorithms' relative performance two benchmark algorithms were used for comparisons: random sampling, and a standard binary Holland GA. Both are defined below.

Benchmark 1: Random Search: Select a point at random, evaluate its fitness. If the value is better than the current best value, store it.

Benchmark 2: Binary GA: Single pool, one-point crossover, bit-flip mutation, truncated selection.

The test problems were as follows (see Appendix A for full details): **TF1**: Counting Ones (32 bits for the binary representation), **TF2**: Royal Road 1 (32 bits for the binary representation [FoMi93]), **TF3**: Royal Road 2 (32 bits for the binary representation [FoMi93]), **TF4** Shekel's Foxholes (16 bits for the binary representation [DeJo75]), **TF5** Shifted Shekel's Foxholes (16 bits

for the binary representation [KiDe95]), **TF6**: Rastigrin's Function (16 bits for the binary representation [MuSV92a]) and **TF7**: 4x4-bit Multiple Deceptive problem [KiDe95], [Sysw93], [Whit91]. The results for each of the algorithms is given in Table 4.4.1.

In Table 4.5.1 each score represents the number of times the algorithm found the *global* optimum out of 30 trials. Table 4.5.1 shows the number of times the global optimum was found for each of the runs, for each algorithm on each test function. It was considered that the global optimum (in conjunction with a fixed sampling rate) was a simple performance measure that should give a reasonable measure as to how well each of the algorithms dealt with a variety of search problems.

Algorithm	Search-Type	TF1	TF2	TF3	TF4	TF5	TF6	TF7	Av.
Benchmarks	Random	0	0	0	13	5	10	2	4 3
	Binary GA	30	1	0	6	7	17	0	8.7
Multi-Rep	DRGA	27	8	2	12	14	30	16	15 6
GAs	Diffusion	30	1	0	7	8	21	0	9.6
	Island 1	19	0	0	12	19	30	9	12 7
	Island 2	0	1	0	23	11	30	19	12 0
	Seasonal	0	0	0	5	11	30	5	7.3

Table 4.4.1: GAs using multiple representations on the test fitness functions (scores out of 30).

As can be seen the random sampling does worst, despite the high sampling rate for TF4, TF5, TF6 and TF7. Overall, the MR-GA in the form of the DRGA does best with an average hit ratio of 15.6. Moreover, all of the MR-GAs, with the exception of the seasonal MR-GA outperforms both random sampling and the standard binary GA.

What is also interesting to note is the that DRGA is the only algorithm to score for all seven problems, and it used the highest number of representations in the course of a run. The DRGA differed from the other MR-GAs in that it used a transmutation operator in conjunction with a heterogeneous population (in terms of representations).

The advantage that the MR-GAs offer (certainly for this set of problems) appears to be related to the ability of the algorithm to apply multiple search strategies to the given problem. The DRGA is the best example of this approach in that all members of the population have potentially different representations, and crossover between individuals is dictated by the individual with the highest fitness. This may suggest that an individual with high fitness is using a representation that is advantageous for the given problem, and that this information is shared via crossover with less fit individuals.

The other MR-GAs in contrast have relatively static forms of representation in comparison with the DRGA, and this may impose an unnecessary bias in terms of the search trajectory. Drawing general conclusions form this small set of test functions is however, extremely hard. For example, it is known that over the universe of possible functions all algorithms will perform equally well if judged in terms of finding the global optimum [WoMc95]. However, what the DRGA does offer is the potential to counter some of the problems associated with designing a good representation for a given problem, and on the basis of the above search problems is a good candidate algorithm for cases where little is known about a correct representation. On this basis the DRGA will be used throughout the remainder of this thesis. In a later chapter we shall test

this decision by running some further comparisons between the DRGA and the standard Holland GA for financial time series analysis problems.

4.5 Summary

In this chapter we have provided a detailed analysis of the GA search heuristic. In this chapter we have concentrated on the core issues that drive the GA search, in an attempt to distil what factors are influential in terms of achieving a good GA performance. One aspect that has become particularly evident is the role played by the choice of representation. Representation has been shown to be crucial to the underlying search mechanism that guides the GA, and that the issue of good representation is at the heart of developing good GA solutions. To tackle this problem in a broad way a strategy of multiple representations has been developed, which has been empirically verified on a small class of test problems.

What the preceding sections have also illustrated is some of the practical manifestations of search equivalence. In essence this gives rise to the fact that the relative difficulty of a fitness landscape is algorithm dependent. We have shown that rather than a complex remapping of space, a simple base change can be sufficient to change the comparative difficulty of a search problem. This underlines the fact that in the absence of *a priori* knowledge about the function landscape (as defined by any algorithm), the choice of operator or representation (real-valued, binary, Gray-binary, non-standard) is completely open, with no formal means for determining the correct choice.

However, what has also emerged from this discussion is the relative ease by which a function landscape can be reshaped by changing operators or encodings. This provides a simple way for employing multiple search strategies to a given problem by dynamically adjusting the algorithm's view of the function landscape. Moreover, in light of the limitations on all forms of search, a random sampling of search heuristics applied to a given problem provides an extremely pragmatic means for probing the space of possible search strategies that may unlock a given search problem. Before describing the methods a legitimate objection might be: if all search strategies are equivalent why use one at all? i.e., use random sampling. There are two immediate reasons. Firstly, little is known about the actual complexity of most real-world search problems, certainly not enough to conclude that in the absence of good partial knowledge of a problem type that search heuristics should be abandoned. Secondly, many researchers have experience of tackling real-world problems with existing search methods and have achieved good results (certainly compared to random sampling). On this basis techniques that enhance existing search methodology should be encouraged. In this sense multiple search heuristics have a pragmatic quality.

However, as yet it is still unclear as to whether the use of multiple representations will solve the neural network optimisation problem, and that two factors look to combine to make this problem particularly difficult. Both issues relate to the sensitivity of the techniques that we wish to combine. It has been shown that GAs are sensitive to representation – a good representation performs well, whereas a poor representation can perform badly. In Chapter 3 it was shown that

neural networks are also sensitive to design factors. It was shown empirically that a neural network performance could hinge on a matter of small changes in the scaling of the data, or on the addition of a few extra training cycles [Chapter 3]. This suggests that unless careful safeguards are taken, the direct combination of GAs to neural network parameterisation could overfit the training set. Having reviewed and investigated both neural networks and GAs in the next chapter we set about formulating the strategy for combining the techniques in order to automate a neural network time series modelling application, and investigate more closely the specifics for constructing NN-GA hybrids.

Chapter 5

Hypothesising Neural Nets

This chapter introduces a technique for empirically testing feed-forword Neural Network architectures. The technique, Artificial Network Generation (ANG), makes possible a controlled series of experiments that statistically validates Occam's Razor as a design methodology for network architectures in the context of gradient descent learning algorithms. This chapter introduces a new method for network architecture pruning, Network Regression Pruning, (NRP). NRP differs radically from existing pruning algorithms in that it attempts to hold a trained network's mapping fixed as the pruning procedure is carried out. ANG is used to analyse the pruning technique's ability to deliver Minimally Descriptive Networks. A method is introduced that uses NRP to infer a small set of candidate architectures for a given learning problem. Finally, it is shown how NRP can be used in conjunction with a Genetic Algorithm for full Neural Network parameterisation.

5.1 System Objectives

In the two preceding chapters a detailed investigation has been conducted into both Neural Nets (NNs) and Genetic Algorithms (GAs). In this chapter we return to the overall objective of designing an automated system for time series analysis. One of the system's main goals is the ability to automatically hypothesise, test and validate neural network models for a target time series problem. In this chapter we concentrate on devising methods for automating the neural network design process. As a statement of objectives, an automated system should have the means to deal with the neural network parameter choices that were presented in Chapter 3 - Table 3.1. One of the first, and most important issues involved in the design of a neural network is the specification of the network's architecture. At present the relationship between the neural network architecture and the neural network performance in terms of generalisation is very open, and no formal relationship has been derived that relates an optimal neural network architecture for a specified learning problem. Results that do exist relate to linear threshold activation functions [BaHa89] and do not take into account continuous activation functions at the hidden and output layers.

There are a number of fundamental problems which are prohibitive in tackling this issue in a formal sense, not least the fact that three-layer feed-forward neural networks have been shown to be universal functional approximators. This implies that in principle a larger network is always capable of approximating the mapping found by a smaller network. This further implies that minimising the number of parameters in a network architecture does not *a priori* lead to better generalisation, and that therefore the advantage in using minimal complexity in the network design has yet to be proven. Moreover, the difficulty in establishing a formal resolution to this problem is compounded by the training algorithm. That is to say, we know that a three-layer feed-forward neural network using sigmoid activation functions is a universal functional approximator

and that therefore, in principle, it is always possible to show that there exists a mapping for a larger network that approximates the generalisation performance of a smaller network. It is therefore imperative in any investigation of the generalisation ability of different network architectures that the method by which the weights are derived is included. That is, it is only at the training phase that two network architectures have the potential to find different network mappings that lead to different network generalisation performance. Moreover, because a network starts from random weight settings any result we can achieve in demonstrating that smaller networks out-perform larger networks in terms of generalisation will be probabilistic. That is to say, it is always possible that certain weight settings may lead a larger network to out-perform a smaller network on a given learning trial.

The first task of this chapter is to devise empirical methods for tackling the above so that an estimate of the probability of a smaller network out-performing a larger network on a given learning trial can be made. If we can establish this it then becomes possible to infer neural network design principles for arbitrary learning problems. Such principles can then be used as the basis for an automated neural network design system.

5.2 Hypothesising Neural Network Models

We are concerned with generating neural network models that can generalise. For time series analysis this equates to whether the trained net can make good out-of-sample forecasts. One approach is to consider the space of all neural network models for a given time series problem. However, as described in Chapter 3, even for small networks this space is too large for a direct search.

One option might be to use a GA to search the space of models by generating populations of neural networks and selecting the best on the basis of generalisation. However this direct approach is extremely expensive in terms of the computational resource required to train and test multiple networks, and if generalisation is to form the basis of the fitness measure it may also prove extremely expensive in terms of the data required (for example if jacknifing or V-fold validation techniques are used).

Another problem with a direct GA-NN approach is the issue of representation [HaSa91], [ScWE90], [GrWi93], particularly with respect to neural network architectures. Chapter 4 gave a good demonstration of how sensitive a GA's performance can be in relationship to the representation, while in Chapter 3 it was shown that a neural network is sensitive to its parameter choices. This suggests that unless great care is taken in the design of a GA-NN system an extremely brittle result could emerge.

Ideally, we should like to investigate the possibility of restricting the size of the search space from all possible neural networks to something more manageable in terms of a GA search. Preferably, a limited set of good candidate neural networks architectures should be hypothesised on the basis of in-sample data so that a GA can then be used as a means of fine tuning parameters and validation. One approach to this issue is to use an existing heuristic.

5.3 Occam's Razor and Network Architecture

To date one of the most commonly used heuristics to ensure good generalisation has been the application of Occam's Razor [Chapter 3]. For feed-forward neural networks it is has long been conjectured that the number of free parameters in a network and the network's ability to generalise are crucially linked, and that in order to promote good generalisation some form of bias towards less complex models is vital. In practice, for supervised neural network learning, this principle is best illustrated by the introduction of various forms of regulisers, or pruning strategies [Chapter 3]. In the extreme form Occam's Razor produces a Minimally Descriptive Model (MDM), thus a model that has the smallest possible complexity[1], and yet can still describe the data set. The first question is how to go about formulating *and* testing a neural network design strategy based on Occam's Razor. We come back to this question in section 5.4, after we have reviewed some existing architecture selection methods.

5.3.1 Existing Regulisation and Pruning Methods

One approach that has been taken for designing network architecture's is to start with what may be an overly large network and then use some means for determining simplification either via pruning, or via post-training weight analysis. As mentioned in Chapter 3 the most common form of regulisation is the use of a weight decay term in conjunction with the usual cost function [WeHR90]. The idea is to associate a cost with each connection during training. This has the effect that both the usual cost function and the reguliser term are subject to minimisation during training. Weights with a zero value are taken to imply no connection. Two common examples of weight decay are the ridge regression form, (here $\mu\sum_{i=1}^{W} w_i^2$ is added to the cost function. Where μ is a control multiplier and w_i are the weights), which biases against large weights, and the Rumelhart form [RuHW86] given by $\mu\sum_{i=1}^{N} w_i^2/(w_0^2+w_i^2)$ which penalises intermediate weights near a distinguished set [WeHR90].

Other methods have used post-training analysis of the weights. For example, sensitivity analysis [MoUt92] and Optimal Brain-Damage (OBD) [LeDS90] introduce methods for analysing the weights in order to determine a weight's *sensitivity*, or *saliency* to the training set. The idea is to find a measure of the contribution of a given weight to the overall output of the network. Weights are ranked according to this measure and then systematically removed according to their rating and the network is retrained. This is similar to an earlier technique of *skelitization* [MoSm89] which removes nodes that have the least effect on the output error (essentially weights with small values).

Improvements on these techniques have also been introduced. Optimal Brain Surgery [HaSW92] improves the measure of saliency used in OBD. One of the latest techniques is Principal Component Pruning (PCP) [LeLM94]. Here a trained net is analysed via principle

[1]Complexity is taken as the number of adjustable weights in the network.

component analysis (PCA). PCA is a standard linear analysis tool used for suppressing redundant information. This technique has the advantage that it is applied once to each layer of the trained net, pruning weights layer by layer. The resultant net is not retrained, but rather the pruned net is used directly. In an earlier example de Groot and Wertz [dGrW90] also used PCA in order to determine the number nodes in the input layer for a neural network time series analysis application.

5.3.2 Why use Occam's Razor?

All of the above techniques are motivated by the desire to increase generalisation on the assumption that reduced network complexity improves generalisation. The ability to be able to test this assumption would be of significant benefit. At present all of the above techniques cannot be said to have been validated, in the sense that no formal study has linked the generalisation ability of a network to its architecture. At present the only evidence that exists is the fact that the authors of the above methods have reported better results using their techniques as compared to not using them, but at no stage has a systematic investigation been conducted. The reason that this situation has arisen is the fact that it is extremely hard to formulate a strictly controlled experiment to test a pruning strategy in the absence of known solutions. For example, unless the pruning strategy has some known target (i.e., a known solution architecture) how can the results be judged? Moreover, if it is not possible to test a pruning strategy in an objective sense, what evidence is there that supports the use of Occam's Razor in the first place? Thus strictly speaking before we set out to formulate a network pruning strategy we should first try to answer the question: is it true that neural networks of minimal complexity generalise better that networks of greater complexity? If we can answer this question, at least in an empirical sense, we can then set about formulating a pruning strategy. As a statement of objectives we have the following: firstly is it possible to show that Minimally Descriptive Networks generalise better that networks with higher complexity? If so, then is it possible to develop techniques for finding the Minimally Descriptive Network for a given problem? Over the next few sections we develop techniques for realising these objectives. We start by introducing a method for testing Occam's Razor.

5.4 Testing Occam's Razor

To test Occam's Razor in the context of neural network architecture design means that we must first establish that a neural network's architecture is related to the overall generalisation performance of the trained neural network for a given learning algorithm. One way to do this is to use a set of test problems for which we have known Minimally Descriptive Network solutions. We can then train networks of various complexities and compare the generalisation performance. If we can do this it will then be possible to statistically evaluate whether Minimally Descriptive Nets, as advocated by Occam's Razor, out-perform networks of greater complexity.

The simplest way in which to achieve the above is to *artificially* generate target time series via known neural network models. By doing this we can generate a suite of test learning problems for which we have a known neural network solution (i.e. the topology and weights of the net used to generate the series). It is then possible to formulate and objectively test a range of network

architectures in terms of their generalisation performance on this test suite of problems. In a wider context if we can establish that Minimally Descriptive Networks do out-perform networks of greater complexity we can then use the same suite of test problems as a means for evaluating pruning methods to see which are the most successful at regenerating the target complexity of the generating network.

5.4.1. Generating Time Series

Consider the network N with fixed architecture (with n input nodes, h hidden nodes and a single output node) and fixed weights. If we seed N with the n-dimensional input values $\overline{X}_1 = (x_1, x_2, \cdots, x_n)$, then the response of N can be represented by $N(\overline{X}_1) = x_{n+1}$, where $x_{n+1} = A\left(\theta_0 + \sum_{i=1}^{h} w_i' A(\sum_{j=1}^{n} w_{i,j} x_j + \theta_i)\right)$. Where, $w_{i,j}$ is the weights connecting input unit j with hidden node i and θ_i acts as a bias for node i. w_i' is the weights connecting the hidden node i to the output node, with θ_0 acting as its bias, and A is the activation function (for all subsequent experiments hyperbolic tangent activation functions were used).

If we let $\overline{X}_2 = (x_2, x_3, \cdots, x_n, x_{n+1})$ then for $N(\overline{X}_2)$ we also get a corresponding output x_{n+2}. In this way we can use N to generate a time series X, consisting of the values produced by iterating N on a seeded input vector. This is the usual form of *iterated forecast* for a neural network.

It is therefore possible to generate a time series from any given topology simply by randomly setting the weights, choosing a seed vector and iterating the network's response. However, this technique rarely produces anything like a suitable series. The tendency is for the net to produce a dampened signal, in which the output converges to a single value. Also, randomly generating the weights means there is no control over the type of series that is produced by the network. For example there is no bias towards a Minimally Descriptive Network, and indeed the resultant series might be linear.

To avoid this some method must be devised so as to make sure the target net is at least producing a good non-linear output. In this sense we require a means to ensure that the networks hidden nodes are *working hard* in producing the iterated series. One way to achieve this is to use a GA.

5.4.2. Artificial Network Generation (ANG)

To bias the amount of non-linearity in the target net we introduce a technique we refer to as Artificial Network Generation (ANG). ANG uses a GA to set the weights for the chosen topology. For ANG the GA fitness is a measure of the amount of non-linearity in the series produced by the network. We define the fitness function associated with a candidate weight setting as follows:

$$\text{fitness}(\overline{W}) = \max[\min \sum_{t=0}^{N} (x_t - \sum_{j=1}^{n} \alpha_j x_{t-i} - \beta)^2]. \qquad 5.1$$

Here $\overline{W}$ is the candidate weight settings and N is the number of points in the created time series. We can solve the inner brackets of the above equation by taking the order n regression model for

the series generated by the network. During the GA run, fitness values are assigned according to residual values of the regression model: the higher the sum squared error of the linear regression model the higher the candidate weights are scored. The GA therefore provides a means for searching the space of possible weight values for the generating neural network, so as to produce time series with non-linear characteristics. The mechanism employed by the GA to search weight values is straightforward. Each weight can be encoded and manipulated in the standard way [Chapter 4]. One detail that is important is that the weight ranges used in the GA are restricted from becoming too large. The reasons for this is that the simplest non-linear series the network can produce is a spike, or delta function [Chapter 3], which is wholly inadequate for these experiments. If the weights are left unrestricted the GA quickly finds this solution, and therefore a restriction on weight size is required.

5.4.3 ANG Results

Using ANG several artificial time series were generated. The results produced by the GA tended to be either regular oscillations, or extremely chaotic, with periods of regularity interrupted by both small and large deviations. Five series, generated by five different neural network topologies, where chosen on the basis of how chaotic the series appeared. The five series were used for all subsequent experiments. Four of the series are graphed in Figure 5.4.1, each is identified with the topology of the network used to generate the series.

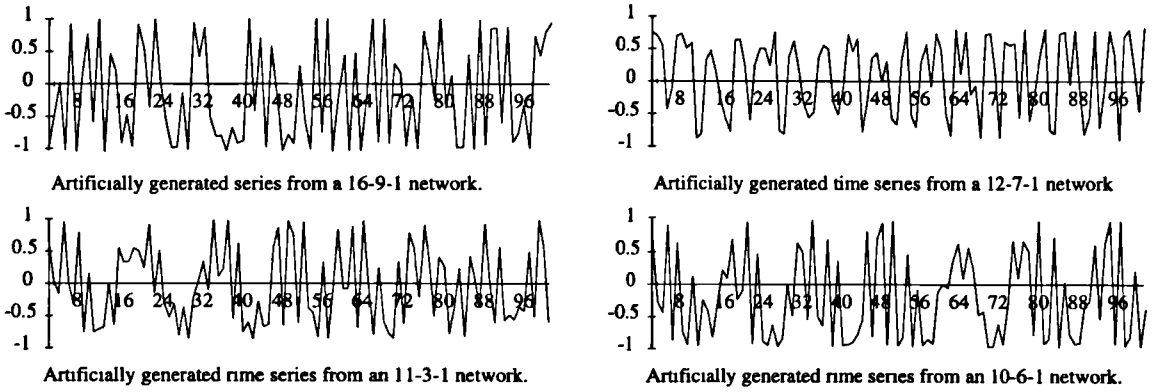


Figure 5.4.1: ANG generated series.

For the artificially generated series to be used as a genuine test of network architectures then the series must be produced by a Minimally Descriptive Network. That is to say, no network of lesser complexity should be capable of generating the series in question. In practice this is very difficult to ensure, as there is no formal mechanism available to guarantee minimal description. However, one way in which to get some measure of the minimal decription of the network is to train networks of lesser complexity and see what level of in-sample training error is attainable. If the series is generated by an approximately Minimally Descriptive Network then we should expect networks of lesser complexity to produce high in-sample training error as compared to the *correct* network topology. In effect, we can use this test as a vetting procedure for the series produced via the GA, with series that show this discrepancy being preferred to those that do not. To test the GA procedure one of the series (generated via a 12-7-1 network) was tested over a

range of smaller network topologies for 30 experiments each. All networks were trained for over 1000 batch weight-up-date learning cycles and started from different random weight settings. The results are graphed in Figure 5.4.2.

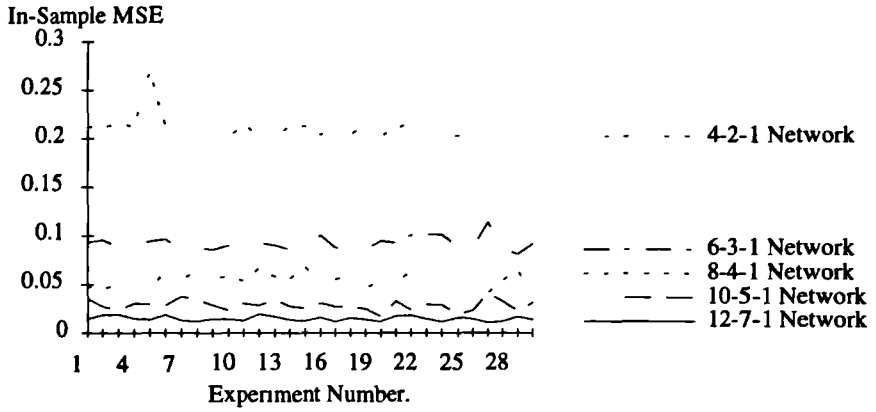


Figure 5.4.2: Networks trained on the 12-7-1 ANG generated series.

As can be seen the 12-7-1 network trained on the 12-7-1 generated series consistently out-performs all of the smaller networks. The worst performance is by the smallest network, with topology 4-2-1, with an average in-sample Mean Square Error (MSE) of 0.211 compared to 12-7-1 network with an average MSE of 0.014. This provides reasonable evidence that the ANG time series creation process can produce the desired approximation to a minimal network complexity.

5.4.4 Testing Architectures

Having created artificial series we are now in a position to test the relationship between network architecture and generalisation. The following experiment was conducted for each of the five test series: firstly a network of the topology of the generating network was trained on 200 points of the generated series. After training, the network made an iterated forecast of 50 steps and the MSE of the forecast versus the target series was logged. This experiment was conducted 30 times using different random weights in the network initialisation phase. This same experiment was then conducted for a new network, but this time the network's topology was made to be significantly larger than the generating network's topology. For training parameters, each network was trained for 1000 batch cycles of the data, with sigmoid activation functions at the hidden layer and a linear output node. The number of batch cycles corresponded with all networks achieving an in-sample MSE of less than 0.01.

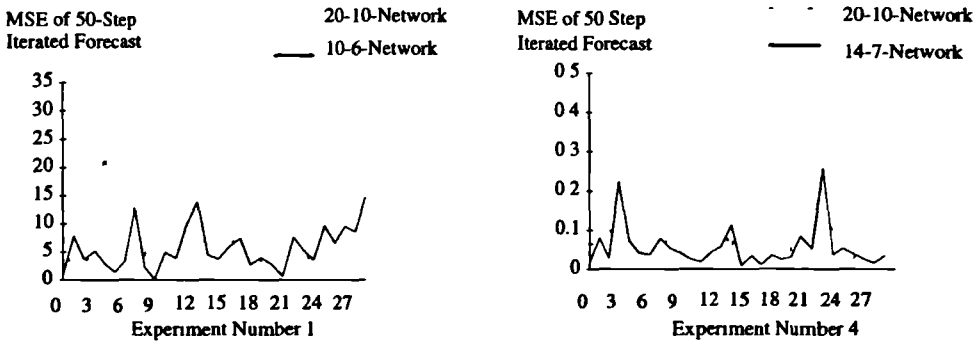


Figure 5.4.3: MSE of iterated forecasts for 10-6-Series and 14-7-Series.

The networks used dynamic learning, with a learning rate multiplier of 1.1 and divisor of 2.0. The data was not scaled or pre-treated. This last aspect was considered unnecessary since the series itself was generated by a neural network.

The results were consistent for each of the five experiments. In each case the network with the *correct* topology out-performed the larger network in terms of the mean MSE of the out-of-sample forecast. Figures 5.4.3 depicts the MSE of the iterated forecasts for two of the series. As can be seen, the results suggest that a more consistent performance in terms of generalisation is achieved by the network that uses the same topology as the generating series. However, what is also evident is the fact that the larger net does not consistently under-perform against the smaller network across all experiments, and therefore only a bias towards smaller nets can be inferred.

In order to fully test all of the experimental results Table 5.4.1 gives the mean MSE and variance of the MSE of the 50-step forecast for each of the five series over the 30 experiments conducted on each.

Experiment Number	ANG Network Topology used to Generate Series	Trained Network Topology	Mean 50 Step MSE forecast (30 trials)	Variance Mean MSE Forecast (30 trials)
1.	10-6-1	10-6-1	5.66	14.41
	10-6-1	20-10-1	6.94	53.15
2.	11-3-1	11-3-1	4.25	5.55
	11-3-1	20-10-1	5.06	1.52
3.	12-7-1	12-7-1	17.128	51.67
	12-7-1	20-10-1	17.982	32.68
4.	14-7-1	14-7-1	0.055	0.002
	14-7-1	20-10-1	0.109	0.157
5.	16-9-1	16-9-1	10.33	6.82
	16-9-1	20-10-1	12.20	11.30

Table 5.4.1: Ten neural network architectures trained on five ANG series.

To gauge the significance of the results the means in all 5 experiments where subjected to standard Z-testing, using the sample means and variances as estimates of the true means and variances. The test statistic used the following:

$$Z = \frac{(\overline{Y}_1 - \overline{Y}_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}, \qquad 5.2$$

where $\overline{Y}_1$ and $\overline{Y}_2$ are the sample means for both networks, and σ_1^2 and σ_2^2 are the respective population variances, and $n_1 = n_2 = 30$ is the sample size. For a two-tailed analysis (using standard normal distribution tables) testing for a difference between means, experiments 2 and 5 were significant at the 90% level (90% and 98% respectively). Experiments 1, 3 and 4 revealed discrepancies at much lower levels (60%, 41% and 54% respectively). For a one-tailed test, testing the hypothesis that the correct topology produces better generalisation (as judged by the MSE of the iterated out-of-sample forecast) the results yield greater than 70% significance for all 5 experiments. These results, in conjunction with the results for the small networks in the previous section, offer some support to the notion of a best-sized topology. However, what is also clear is the fact that larger networks can perform well as compared to smaller networks and that

there may be other advantages in their usage not tested here (for example training times [ZhMu93], [LeDS90]). What is interesting about these results is that despite the fact that larger networks should theoretically encompass the mappings available to smaller networks (for example by setting weights to zero) there is still a high level of *bias* towards better generalisation achievable by using an approximation of a Minimally Descriptive Network.

5.5 Design Strategies using Occam's Razor

The above goes some way in supporting the use of Occam's Razor in the design of a neural network architecture. This still leaves the practical issue of how to derive a Minimally Descriptive Network. As mentioned we are interested in developing a pruning strategy in order to do this.

One important aspect that has not been explicitly considered in the development of existing pruning methods is the question of over-training. That is, conventional pruning techniques attempt to counter over-training but at no stage do they directly address the mapping found by a given trained network in terms of over-training.

Over-training relates to the ability of a candidate model to match too closely the details of a particular training set, to the extent that relationships unique to the training set are incorporated within the model. From this point of view pruning can be seen as a method of limiting the ability of the model to fit arbitrary characteristics of the training data (for example noise). It is therefore necessary when considering a technique for inferring network complexity to take into account some form of equivalence relationship between models. That is, if two networks with wholly different complexities are trained on the same data set, at what limit does it become impossible for a network with reduced complexity to match *approximately* the mapping of the larger network? If a model of vastly simplified complexity can approximately match the mapping produced by an overly complex model, then the reduced model is open to the same level of over-training that the larger model is. This issue goes to the heart of any complexity reducing process in that it suggests that the first stage in finding the target mapping is to understand a candidate mapping.

This step constitutes a radical departure from current neural network pruning methods in that we are suggesting that a network's mapping should be held fixed, while we explore the space of possible topologies. Current practice on the other hand is to explore both spaces simultaneously. For example, the use of a reguliser [WeHR90] trains the network at the same time as it attempts to remove, or surpress, weights. OBD [LeDS90] and skelitisation [MoSm89] removes weights on the basis of their salency and then retrains the network. PCP removes weights layer by layer using principal component analysis, however no analysis of the effect of a weights removal on the mapping found by the network is used. In all of these cases the pruning procedure adjusts the architecture and mapping found by the network. Moreover, at no stage is any attempt made to maintain a network's original mapping.

In contrast to these methods, by attempting to hold a network's mapping fixed, we will be able to place a bound on the size of networks that are prone to over-training and therefore bound

the complexities of networks worth testing. In the following sections we develop this argument and produce a new technique for approximating Minimally Descriptive Network architectures. Before we do this we start with some formal justification for developing a pruning technique that attempts to fix a network's mapping before exploring the space of possible architectures.

5.5.1. Minimally Descriptive Nets

The results of section 5.3.3 suggest that we should favour minimally descriptive solutions. For feed-forward networks, in most instances, we have no way of knowing exactly what the Minimally Descriptive Network should be, and the fact that we are using a stochastic training process (i.e., we start from a random set of weights) means that even if we have the minimal complexity we cannot be sure that the *correct* mapping for a particular problem can be found in a single training run. This implies that in practice for real-world problems it is extremely difficult to make formally accurate statements regarding Minimally Descriptive Networks. However, if for the present we ignore the practical difficulties involved, we can introduce a working definition of Minimally Descriptive Networks as follows: let $C(N_i)$ denote the complexity (number of weights) in network N_i, then:

5.1 Definition: Let g be a learning problem. A training set for g consists of a finite set of input-output instances, $\overline{Y} = \{g(\overline{x}): \overline{x} \in \overline{X}\}$. For any $\varepsilon > 0$, the ***minimally descriptive network*** denoted by N_{mdn} for $\overline{Y}$ is the minimal complexity neural network such that;

$$\sum_{\overline{X}} (g(\overline{x}) - N_{mdn}(\overline{x}))^2 < \varepsilon, \text{ and } \forall N_i \text{ such that } C(N_i) \subset C(N_{mdn}) \text{ we have } \sum_{\overline{X}} (g(\overline{x}) - N_i(\overline{x}))^2 \geq \varepsilon,$$

where N_i is chosen from all networks (over all parameterisations) that have complexity less than N_{mdn}. Note, containment, $\subset$, refers only to the complexity of the networks (i.e., the number of weights) and not the mapping, architecture design or parameterisation. The above simply states that for any given error bound and any given training set the Minimally Descriptive Net (MDN) is the smallest network capable of achieving a fixed in-sample error bound.

In the context of network pruning, definition 5.1 implies that *any* pruning strategy can be used to infer a bound on the complexity of the MDN once a fixed error bound has been imposed. The reason for this is that the pruning procedure can always be applied until the network violates the error bound.

By definition 5.1 all networks prior to error-bound violation must upper-bound the complexity of the MDN. To take this further, definition 5.1 also suggests that what is ultimately important is the fixed error bound as opposed to the actual mapping found by the candidate model. This is important as it suggests that a legitimate procedure for inferring the complexity of the MDN is to fix an arbitrary mapping that satisfies 5.1 and then explore the space of possible architectures capable of approximating this map. This has significant practical implications in that if it is possible to fix a network's mapping then we can explore the space of network architectures by a process akin to *shallowest gradient ascent* i.e., we select network architectures on the basis of minimum disruption to the fixed mapping. In the next few sections we introduce a technique for achieving this. We start by re-examining network training.

5.5.2 Network Model

To recap on the Multi-Layer Perception (MLP) model under investigation we have the following: a univariate time series problem consists of a time-dependent process X generating a sequence of observations according to, $x_t = g(x_{t-1}, x_t, \ldots, x_{t-n})$, where x_t is the current response (dependent variable), $x_{t-1}, x_{t-2} \ldots x_{t-n}$ are past values of the series, and g is the target function to be estimated[2]. As described in Chapter 3 we are interested in constructing an estimate $\hat{g} = f$ for g from a large class of functions F where F is bounded by the architecture of the network. The class of MLP under consideration is given by:

$$\hat{x}_t = A\Big(\theta_0 + \sum_{i=1}^{h} w_i' A\Big(\sum_{j=1}^{n} w_{i,j} x_{t-j} + \theta_i\Big)\Big), \tag{5.3}$$

where, h denotes the number of hidden units in the net, n the number of input units, $w_{i,j}$ the weights connecting input unit j with hidden node i and θ_i acts as a bias for node i. w_i' is the weights connecting the hidden node i to the output node, with θ_0 acting as its bias. The activation function A is the hyperbolic tangent [Chapter 3].

Having defined the output of the net the training phase consists of adjusting the weight vectors to minimise a cost function, $E = C(\hat{x}_t, x_t)$, where E is the error of the net as defined by the cost C. In this instance we take the cost function to be the single distance measure of the squared difference between the network output and the desired output pattern i.e.,

$$E = \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 . \tag{5.4}$$

The parameterisation of equation 5.3 is effected by some form of gradient descent over the weight landscape in order to minimise the cost function for the complete training set [Chapter 3].

5.5.3 Network Regression Pruning (NRP)

Once a network has been trained genuine redundancy in the parameterisation of the model can only be tested if it is possible to reproduce the *exact same mapping* (over the training set) with a reduced network architecture. In practice it is not feasible to find the *exact same mapping* in a formal sense, but it is possible to find an approximation to the candidate mapping by a form of shallowest gradient ascent through the space of possible architectures. That is, each move in architecture space is based on the minimum disruption of the original mapping as judged by the in-sample MSE of the network output.

To see how such a procedure can be defined we have the following. We first start with a large fully connected, one hidden layer feed-forward network using sigmoid activation functions at the hidden layer (hyperbolic tangent) and linear output nodes. Training the net with a standard variation of backpropagation on a target time series for N input/output pairs, $(x_t, x_{t-1}, x_{t-2}, \cdots x_{t-n}, t = 1, \cdots N$, where x_t is the desired output, and $x_{t-1}, \cdots x_{t-n}$ are the n input patterns at time t) means the network has formed an estimate $\tilde{g} = f \in F$ for g.

---

[2]For the present we ignore the effects of noise.

Having trained a network on a particular data set the resultant weight vectors are given by $\overline{W_i}$ for each of the weights from the input nodes to hidden nodes ($1 \le i \le h$, for h hidden nodes), and $\overline{W'}$ for the h weights connecting the hidden layer to the output node. If we note the corresponding input for each node for each of the N input/output patterns of the training set we get the following relationships,

$$\alpha_{i,t} = \overline{W_i}.\overline{X_t} + \theta_i \text{ and } \hat{x}_t = \overline{W'}.\overline{Y_t} + \theta_0 \qquad 5.5$$

for input-to-hidden nodes and hidden-to-output nodes respectively. $\overline{X_t}$ is the n-dimensional input vector at time t, and $\overline{Y_t}$ is the corresponding h-dimensional activation vector produced at the hidden nodes. The θ terms being each of the respective nodes' bias input.

The next step in testing for redundancy is to remove weights, but at the same time try to restore the mapping that had been found by the trained network. The purpose of this is to try and see at what level of complexity it becomes impossible to match the mapping found by the original network. The hypothesis is that if the original network is over-trained, then the reduced network will also be over-trained if it can approximate the same values given in 5.5.

From 5.5 it is quite straightforward to see that if a weight is removed the original mapping may be recovered (approximately) by solving for new weight values to the node affected over the whole training set. Thus we use the values given in 5.5 as the target values which must be solved to give the new set of weights. Therefore if the first weight removed connects the k'th input node to the i'th hidden node we need to solve the following for $\tilde{w}_j$ and $\tilde{\theta}_i$,

$$\min \sum_{t=0}^{N} (\alpha_{i,t} - \sum_{\substack{j=1 \\ j \neq k}}^{n} \tilde{w}_j x_{t-i} - \tilde{\theta}_i)^2 \qquad 5.6$$

where $\tilde{w}_j$ are the new values of the weights connecting the i'th node to the input nodes, and $\tilde{\theta}_i$ is the new bias value. Equation 5.6 can be solved by linear regression. How well the new mapping approximates the old mapping is clearly dependent on the residual values of the linear regression process used to solve 5.6.

The above means we now have a procedure for removing weights whilst approximating the original mapping found by the trained neural network. It means we should expect some gradual decay of network in-sample performance as pruning proceeds. The hypothesis is that the error profile of this procedure should provide some clue as to the MDN required for the training problem.

To complete the procedure each weight of the net should be removed according to the order of least in-sample error disruption. The reason for this is that at all stages we want to preserve the original neural network mapping as far as is possible and effect a shallowest gradient ascent over the in-sample MSE-to-architecture surface. At each stage the weights connected to nodes affected by the weight removed can be re-set by linear regression to approximate their previous values, and this can be done exhaustively. The end result will be an in-sample error profile which may provide some clue as to the redundancy in the original neural network mapping, and offer some bound on the number of weights required to approximate the over-trained network's performance.

To summarise the pruning procedure that we refer to as - ***Network Regression Pruning*** (NRP):

Step 1: A large network is trained using supervised learning (e.g., backpropagtion) on a target time series. The network is trained until the in-sample error is extremely small (as measured by the MSE), so as to produce a possibly over-trained network. Call this network N_1.

Step 2: The input values to each node of N_1 is recorded for the complete training set. For node i we represent these values by $\overline{\alpha}_i$.

Step 3: Remove a single weight from N_1, say w_i.

Step 4: Re-set the weights to the node affected by Step 3 using linear regression with the target values given by $\overline{\alpha}_i$ for that node (in the case of a singular matrix use single value decomposition). Call the new network N'.

Step 5: Calculate steps 3 and 4 for each weight in the network. Remove the weight that minimises the difference between the new network and the original network over the complete training set i.e., $\min \sum_{t=1}^{N} (N_1(\overline{X}_t) - N'(\overline{X}_t))^2$.

Step 6: Call the network chosen in Step 5 N_1 and repeat Steps 2 to 6 until all weights have been removed.

The above can be seen as forming new networks by approximating the functionality of the previous network. We should expect that a large number of weights can be removed, without affecting the network's ability to perform its original mapping, and thus remain in a potentially over-trained state. However at some stage we should also expect that the in-sample MSE should start to rise once a threshold of network complexity has been breached. To test the procedure and the underlying hypothesis the next phase is to test the method on series that have known neural network solutions. This we can do by using the ANG series that were generated in section 5.4 as a means of testing Occam's Razor.

5.5.4 Results of NRP on ANG Series

The five artificially created series described in section 5.4 were used as a test suite of problems. To start the experiments a fully connected net of 20 input nodes, 10 hidden nodes and a single output node was trained on 200 points of each of the generated series. This network topology is much lager than any of the generating networks and therefore should, in principle, be able to find an approximation to each of the series, or to simply overfit the training data.

The 20-10-1 networks were trained on each of the series until the in-sample error was of the order 0.01 (this was chosen as being a reasonable bound on the in-sample error). The training parameters were the same as those used in section 5.4. As was expected the networks appeared to over-train and the iterated forecast for each of the series was poor.

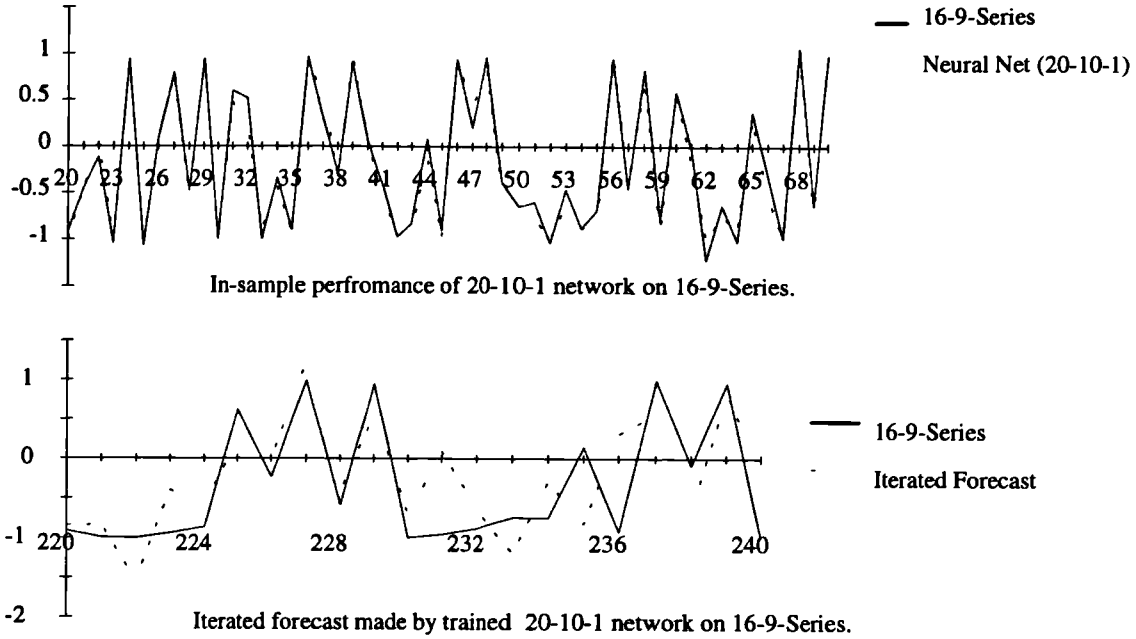


Figure 5.5.1: Iterated forecast by 20-10-1 network for 16-9-Series.

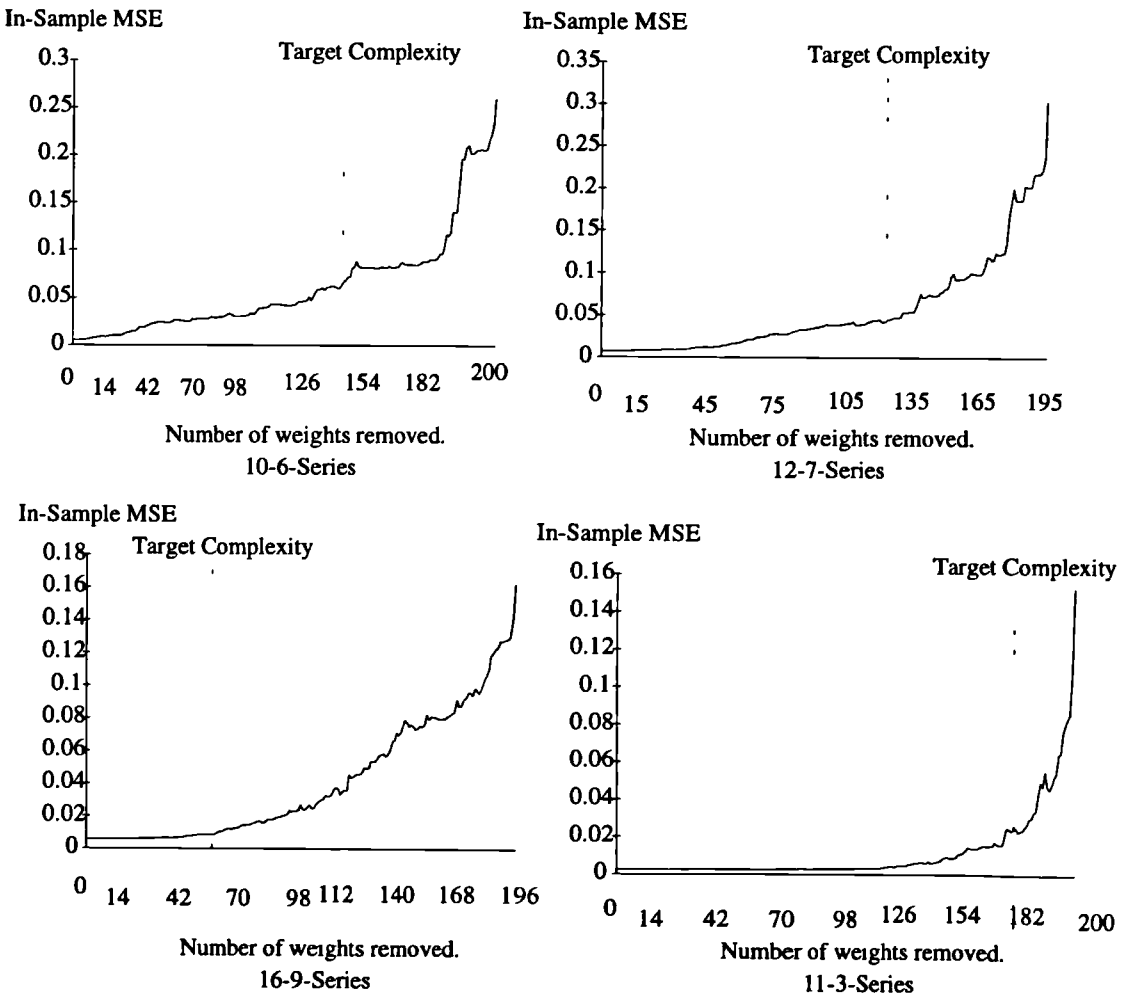


Figure 5.5.2: NRP applied to the ANG generated test series.

Figure 5.5.1 shows both the in-sample fits and iterated forecasts for one of the experiments. This result was typical and suggested that the mapping found by the trained neural network did not correspond to the target mapping.

After training, weights were removed from each of the 20-10-1 networks according to NRP. The results achieved are graphed in Figure 5.5.2 and Figure 5.5.3. As can be seen, as the number of weights removed increases, the in-sample error rises.

In all of the Pruning Error Profiles given in Figures 5.5.2 and 5.5.3 the target complexity of the generating network produced via ANG is marked by a dotted line. The least characteristic of the curves is for the 14-7-series given in 5.5.3. Here the pruning profile is marked by a sharp increase at the very end of the pruning phase.

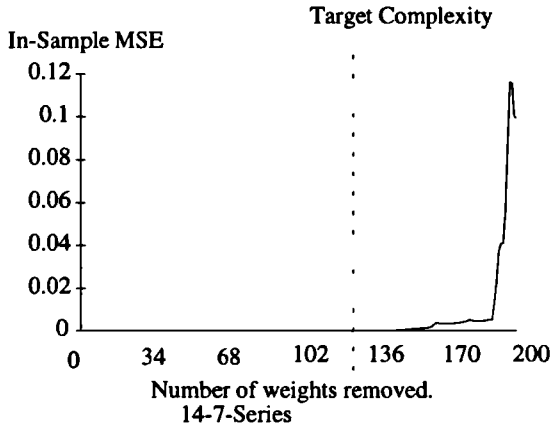


Figure 5.5.3: NRP applied to the 14-7-ANG generated series.

What is interesting to note about Figure 5.5.3 is that in Table 5.4.1 the 14-7-series showed the least sensitivity in switching from a 20-10-network to a 14-7-network, the mean MSE of the iterated forecast being 0.199 and 0.055 respectively, compared to the next least sensitive given by the 11-3-series with a mean MSE iterated forecast of 4.25 for the 11-3-1 network and 5.06 for the 20-10-1. This may suggest that the 14-7-series is near linear, and therefore even much smaller networks than 14-7 could approximate the series. This may be the case if ANG process was stopped too early.

5.5.5 Interpretation of the Pruning Error Profiles

At this stage it is worth re-iterating the purpose for developing a network pruning procedure. The desire is to automate the application of a feed-forward neural network for a time series application. Section 5.4 presented evidence in favour of MDNs in terms of their generalisation ability. In sections 5.5 to 5.5.4 we have derived a method for exhaustively pruning a network on the basis of architectural redundancy. The final step in automating the network architecture design is the ability to infer an architecture for a given learning problem. At present all existing network architecture pruning strategies (reviewed in section 5.3.1) involve a judgmental decision on behalf on the network designer as to how pruning should be controlled. For example, in [WeHR92] the amount of pruning that takes place is controlled by the human network designer and takes into account the *relative difficulty* of the learning problem (see Appendix in

[WeHR92]). Other techniques, such as skeletonization [MoSm89] the human designer must judge the size of the weights and decide what constitutes a small weight (so that it can be removed), or conversely a measure of *saliency* is made and weights are removed either according to a human designer [MoUt92] or on the basis of a validation set [LeLM94]. For a completely automated system a fixed decision process is required in order to determine the network architecture and therefore decisions that involve judgmental measures are inadequate. We would also like to limit the number of network parameters that are set according to a validation set performance. Some of the problems that can arise from this were described in Chapter 3. One new way to tackle these problems is to use Artficial Network Generation (ANG).

ANG raises the possibility of inferring an architecture selection procedure on the basis of the pruning error profiles provided by NRP. That is to say, we can use the error pruning profiles given in Figures 5.5.2 and 5.5.3 to infer a procedure that identifies the target complexities marked on each of the figures given in Figures 5.5.2 and 5.5.3. With respect to the comments made in section 5.4.1 regarding MDN, if the in-sample MSE pruning profile does reveal characteristics of the MDN's complexity then it should be possible to devise a simple threshold rule to act as a trigger for an approximate MDN complexity.

The simplest rule would take into account the rise in in-sample MSE as each weight is removed. We should expect that large rises in the MSE would signal the new network's inability to match the target mapping (in accordance with definition 5.1).

A first rule could therefore use the variance of the in-sample Mean Squared Difference Error (MDSE) as a trigger for significant rises in the pruning error profile. Explicitly we can use the following:

$$\text{Complexity Identifier} = \text{Min}[i \text{ such that } (e_i - e_{i-1})^2 - MSDE > \text{var}(SDE)], \qquad 5.7$$

where i is the number of weights removed, $MDSE$ is the mean squared difference error, var(SDE) is the variance of the squared differenced error, and e_i is the MSE for the network resulting from i'th application of NRP (i.e., removal and regression re-training after the i'th weight is removed). Using 5.7 we get the following identification of network complexities for the five experiments:

Experiment Number	Target Network	Target C mplexity	Suggested Complexity	Error
1.	10-6-1	73	91	18
2.	11-3 1	40	52	12
3.	12-7-1	99	80	19
4.	14-7-1	113	34	74
5.	16-9-1	163	122	41

Table 5.5.1: Complexity Identify used for the 5 ANG series using Eq. 5.7.

The results given in Table 5.5.1 are reasonable if we consider that in all cases bar Experiment 4 the trigger identifies the correct complexity to within 2 hidden nodes of the original 20-10-1 network used before pruning. The 14-7 series has already been singled out as being possibly linear and this may explain why the complexity trigger suggests that a much smaller network is capable of learning the series. An improvement on 5.7 is possible if we take into account the

mean of the squared difference error of all weights removed up to but not including the weight about to be removed, i.e., $msde_j = \frac{1}{j}\sum_{i=1}^{j}(e_i - e_{(i-1)})^2$ and apply a dual threshold to the complexity trigger. The reason that a dual threshold is applied when judging the significance of the MDSE is that we are interested in the point at which a network becomes incapable of approximating the original mapping found by the 20-10-1 network. We can therefore assume that a lower threshold will mark the boundary of statistically significant rises in the error profile, and an upper threshold will exclude the increases in error caused by an already catastrophically failing network. On this basis we can infer a trigger of the following form:

$$\text{Complexity Identifier} = \frac{1}{n}\sum_{i=1}^{N} i.\chi(e_i), \text{ for n} = \sum_{i=1}^{N}\chi(e_i), \qquad 5.8$$

where,

$$\chi(e_i) = \begin{cases} 1 \text{ if } k_1 \text{ var}(sde_i) < d(e_i, e_{i-1}) - msde_i < k_2 \text{ var}(sde_i), \\ 0 \text{ otherwise.} \end{cases} \qquad 5.9$$

and $d(e_i, e_{i-1}) = (e_i - e_{i-1})^2$, and sde_i is the squared difference error up to weight i. Using the five series the optimised values of k_1 and k_2 give k_1=5.6 and k_2=7.1, with the best results achieved by restricting n to the top 5 violations of 5.9.

Experiment Number	Target Network	Target Complexity	Suggested Complexity	Error
1.	10-6-1	73	79.8	6.8
2.	11-3-1	40	40.2	0.2
3.	12-7-1	99	108.8	9.8
4.	14-7-1	113	67.8	45
5.	16-9-1	163	147.8	15.2

Table 5.5.2: Complexity Identifier (Eqn. 5.8) used for the 5 ANG series.

Table 5.5.2 suggests that equation 5.8 can be used as a reasonably effective means for bounding the complexity of a network architecture for a given learning problem. In all cases bar Experiment 4 the error of the predicted complexity is less than 16 weights, which in terms of the 20-10-1 neural networks used as the starting architecture translates into an error of less than 1 hidden node in determining the *correct* complexity.

The relatively poor result for Experiment 4 may, as discussed, be due to the possible linearity of the 14-7-series. On this basis the network complexity identifier given above seems to be a workable starting point for automating the design of the network topology.

5.5.6 Determining Topologies

Having established a method for inferring an appropriate network complexity the next step involves the determination of an actual network topology. What we desire is an automated mechanism for turning a target complexity into a specific 3-layer neural network topology. Testing all possible network configurations that produce a particular network complexity is impractical for a fixed computational resource. One method might be to use the topology that corresponds to the target complexity (as given by equation. 5.8) during NRP. However, equation

5.8 involves an average of several complexities, which may be contradictory in terms of the actual architectures suggested. Moreover, this approach would put a large emphasis on the accuracy of the pruning procedure, which may not be justified. That is to say, bounding a complexity requires less precision than determining a precise architecture. However, no attempt will be made to clarify this issue at present, and such a study will be the subject of future work. An alternative is to test for fully connected networks of the same complexity as the target complexity, bounded by the size of the pre-pruning network. To demonstrate this strategy we have the following for Experiment 1 above (see Table 5.4 1). For Experiment 1 (target topology 10-6-1) the suggested complexity of the network revealed by the pruning process was 79.8. Translating this into fully connected networks bounded by the original 20-10-1 network gives the following possible configurations for testing based on a target complexity of 79.8.

Topology	Complexity	Topology	Complexity	Topology	Complexity
20-4 1	89	15-5-1	86	10-7-1	85
19-4-1	85	14-5-1	81	9-7-1	78
18-4-1	81	13-5-1	76	8-8-1	81
17-4-1	77	12-6-1	85	7-9-1	82
16-4-1	73	11-6-1	79	6 10-1	81

Table 5.5.3: Suggested complexities for Network Architectures for Experiment 1.

Here the complexity is calculated as, $Complexity = In \times Hidden + 2 \times Hidden + 1$. Where *In* is the number of input nodes, and *hidden* is the number of hidden nodes. The right-hand side of the equation includes the network's bias nodes (one for each hidden node and one for the output node). On this basis candidate topologies can be produced using this equation for each fixed value of *In* (within the starting network's bounds) and solving for *hidden* using the pruning target complexity (the value of *Complexity*). The topologies considered in Table 5.5.3 are those that are fully connected and represent the minimal difference between the target and generated complexity. The reason that networks smaller than 6 input nodes do not appear in Table 5.5.3 is that to include 5 input nodes would require a hidden layer of 11 nodes which is outside the range of the starting network (20-10-1). Using the above gives 15 candidate networks (compared with the 200 possible fully connected networks) which is a feasible number to train and test directly.

5.6 Validation

Progress has been made towards automating the design of a neural network for time series analysis. We have developed and tested a technique for hypothesising neural network complexities for arbitrary learning problems (NRP), and have inferred a technique for isolating candidate topologies (Equation 5.8). However, referring to Table 3.2.1 [Chapter 3] there are still a number of free parameters that must be fixed if we are to fully automate the neural network design process.

In particular there is the question of neural network selection in terms of validation and training. That is to say, how are parameters associated with the actual training of the network to be fixed, and how are we to select the final candidate model for a given problem? Specifically we need a means for deciding;

1) Which of the candidate topologies should be used?

2) What learning parameters should be used (i.e., data scaling, learning rate, momentum term, learning style)?

As mentioned in 5.4 candidate topologies can be tested directly if we restrict the candidate networks to being fully connected. However, this still leaves open the actual basis for deciding how one network's performance should be preferred to another. The same issue arises with regard to deciding learning parameters.

Having restricted the search space of possible networks down to i) a small number of candidate topologies, and ii) the fine tuning of training parameters, it seems reasonable at this stage to introduce a Genetic Algorithm (GA) to make the final selection of the remaining free parameters. We briefly review some of the existing methods for combining GAs and neural networks before we present our chosen method.

5.7 GA-NN Hybrids: Representations

In an attempt to solve some of the neural network parameterisation problems there has been considerable research effort into combining neural networks and GAs. GAs have been used for weight training for supervised learning [ScWE90], [BeMS90], [MoDa89], for selecting data [ChLi91], setting training parameters [BeMS90], and designing network architectures [MiTH89], [WhSB89], [WhSB89], [HaSa91].

The network design problem can be seen as a search for an architecture that best fits a specified task according to some explicit fitness measure. Before we review fitness measures, one of the first priorities for GA usage is a representation of the problem. Neural network representation for GAs has been tackled from many directions with varying degrees of abstraction. The most general methods fall into three categories [Grua93]:

Direct Encoding: A graph data structure of the neural network is encoded and manipulated via the GA. An example of this is given in Whitley et al [WhBo90], [WhSB89]. An n^2 (where n is the number of nodes in the network) connection matrix is used as a chromosome or individual. The GA can either use standard neural network training and then apply a fitness measure, or manipulate weight values at the same time as the connectivity (this may lead to larger matrix representations depending on the encoding of the weights). Belew et al [BeMS90] use a combined approach, in which the GA is used to design the net, set good initial weight values and then use backpropagation to complete training.

Parameterised Encoding: A list of parameters is designed to describe the network. For example, parameters code the number of layers, the size of layers, and the connections. Here the GA manipulates a list of parameters rather than the structure itself (as in direct encoding). The parameters that are included can range from training parameters for learning algorithms all the way through to weights themselves. Harp and Samad [HaSa91] give an example of this, coding the properties of the architecture and the learning parameters for neural network training in one long binary bit string.

Grammatical Encoding: A graph generation grammar is created that encodes the network configuration details. The GA then manipulates the grammar, and in most cases applies the grammar to construct the net and then uses a form of neural network training to fix the weights [GrWh93].

For the representations presented above the option whether to use neural network training or to use the GA to set the weights is left open. More abstract approaches have neural network training in-built in the representation. For example, Koza has introduced Genetic Programming [Koza94] which directly searches function space (symbolic regression), replacing the need for any parameterisation of a fixed model (as in the weight training for a fixed neural network architecture). A less direct approach, in the sense that it retains features of neural network training, has been used with grammatical encodings. Gruau [Grua93] and Zhang [ZhMu93] have both introduced means for directly setting weights and architectures as part of neural network generating grammars. The representation is clearly a vital issue in this form of GA-NN combination, but as has been made apparent in Chapter 4, the representation is inherently linked to the overall objective of the GA search. Representations can only be judged in terms of what the GA is attempting to solve. This brings us on to fitness definitions.

5.7.1 Fitness Measures for GA-NN Hybrids

One surprising fact is that all of the GA-NN hybrids mentioned so far have used fitness measures based on the computational performance of the neural network rather than focus on the problem solving characteristics of the neural networks created. For example, the most widespread fitness measure is training time and network complexity [GrWh93]. The reason that this is surprising is that the goal for neural network design is generalisation. One reason that may account for this apparent lack of emphasis on generalisation is the difficulty in defining good generalisation metrics, and the inherent danger of optimisation over such a metric. As was discussed in Chapter 3 metrics for generalisation, particularly for time series analysis, have largely been based on validation sets. If a GA (or indeed any automated control system) is introduced into both the validation and the training cycle then over-training seems likely.

Another problem is that if neural network generalisation is to be used as the fitness measure for a GA then the representation issue becomes even more crucial. As has been mentioned, generalisation for neural networks is extremely sensitive to training parameters, and in turn GAs are extremely sensitive to representation. This suggests that to combine the methodologies in a direct way runs the risk of producing the worst of both worlds.

5.7.2 Neural Networks and GAs: Fitness Measure for Generalisation

Generalisation has not featured as a fitness measure for GA-NN hybrids and there is very little in the standard literature for GA-NN combinations for time series analysis. In light of the problems associated with generalisation and neural network design this thesis will adopt a fairly cautious approach in terms of combining GAs and neural networks. To this end, this task has been made easier by the techniques that have already been introduced in this chapter. NRP provides a means for inferring a small number of candidate neural network architectures for a

given learning problem. This implies that NRP circumvents the need for a complex grammatical encoding of the neural network for the GA. It therefore seems reasonable that the GA can be used to optimise the learning parameters for a small number of network architectures supplied by NRP. To do this we will require a fitness function. As stressed above we are interested in generalisation. For neural networks most researchers employ some form of multiple cross-validation [Chapter 3]. In line with the discussions of Chapter 3 for this thesis a composite error measure can be defined as follows:

Let D be the full data set. D is divided into v data sets of equal size. Let H be the iterated forecast horizon for net N_i. For each step of the iterated forecast of net N_i let x_t and $\hat{x}_t$ be the target output and neural network output respectively. For each data set j $(1 \le j \le v)$ we form two error measures:

$$E_{j_1} = \frac{1}{H}\sum_{t=1}^{H}(x_t - \hat{x}_t)^2, \tag{5.9}$$

$$E_{j_2} = \frac{1}{(H-1)}\sum_{t=2}^{H}((x_t - x_{t-1}) - (\hat{x}_t - \hat{x}_{t-1}))^2, \tag{5.10}$$

where E_{j_1} is the mean square error of the iterated forecast over the forecast horizon, and E_{j_2} is the mean square error of the 1-step gradient of the iterated forecast. The idea behind E_{j_2} is that the direction of the forecast is of equal importance as the magnitude of error, this is of particular significance for financial forecasting where we are interested in forecasting financial trends (price rise or fall). The final error measure is taken over v giving,

$$E = \frac{1}{v}\sum_{j=1}^{v}(E_{j_1} + E_{j_2}). \tag{5.11}$$

In most cases v is chosen to be between 5 and 10 [MoUt91]. The task of the GA is to minimise Equation 5.11 over four main parameters: the topology, the learning rate multiplier, the learning rate divider, and data scaling. The topologies available to the GA are fixed by NRP (i.e., for Experiment 1 the GA has a choice of 15 candidate topologies). The learning rate, λ, for each network refers to the learning rate multiplier and learning rate divider for each weight update [Chapter 3]. Both multiplier and divider are restricted to the range 1 and 5 with the GA acting to a resolution of 0.01 [Chapter 4]. The data scaling is restricted between values ± 0.5 and ± 0.9 (for the training data regardless of the values of the validation set) again with the GA acting at a resolution of 0.01. The remainder of the neural network parameters were held fixed, this included hyperbolic tangent activation functions for the hidden layer, linear output nodes, batch weight update and a fixed number of training cycles. The results of the GA-neural network combination will be explored in Chapter 7 when we discuss the target financial series that is to be modelled.

5.8 Summary

The purpose of this chapter was to lay the foundations for the design of an automated system capable of applying feed-forward neural networks to arbitrary time series problems. The principal objective of such a system is to infer a neural network design that increases the likelihood of

good generalisation. To this end this chapter has introduced three techniques that aid the neural network design process. Firstly we introduced the Automated Network Generation (ANG) procedure. ANG provided a means for testing neural network design methodologies within a controlled environment. It was used to test the hypothesis that Minimally Descriptive Networks (MDNs) have an increased probability of good generalisation as compared to larger network designs. We established empirical evidence in favour of Occam's Razor and suggested that MDNs should be the target of a neural network design process. We then introduced a method for inferring MDNs via network pruning. Network Regression Pruning (NRP) is substantially different from existing pruning techniques in that it explores the space of possible architectures whilst attempting to hold a network's mapping fixed. This strategy was then combined with ANG so as to infer a network architecture identification procedure (Equation 5.8). This procedure used a simple statistical analysis of the pruning error curve generated via NRP to isolate a candidate network architecture. Finally, it was suggested that the remaining network parameters (such as data scaling and learning rates) can be optimised via a GA. The results of this chapter will be used in Chapter 6 in the design of an Automated Neural network Time Series Analysis system (ANTAS). Once the design of ANTAS is completed in Chapter 7 we shall apply and present the results of the system for a financial time series problem.

Chapter 6

Automating Neural Net Time Series Analysis

This Chapter presents the design of ANTAS (Automated Neural net Time series Analysis System). ANTAS combines feed-forward Neural Networks (NNs) and Genetic Algorithms (GAs) so as to automate a Neural Network time series modelling application for an arbitrary time series.

6.1 System Objectives

In this chapter we combine the results and analysis of Chapters 3, 4, and 5 in order to present the design of an automated time series analysis system, ANTAS. The primary objective for ANTAS is the ability to automatically hypothesise, test and validate predictive models for a target financial time series problem. In terms of financial analysis this problem can be broken down into two levels of complexity. Firstly, the system should have the ability to construct univariate time series models based on a single target series. This we shall refer to as *primary modelling*. Secondly, the system should also have the ability to construct more complex models in which a primary model is extended by the use of secondary data that is possibly influential, or explanatory, to the primary series. This we refer to this as *secondary modelling*. In this respect secondary modelling offers the possibility of improving the predictive quality of a primary model by including additional data series that may in some way *explain* the target series. In this sense, secondary modelling refers to multivariate time series analysis.

The system's objectives can be further specified in terms of the following levels of functionality:

Data Selection and Data Treatment: A module within ANTAS must provide some means for manipulating and pre-treating raw data in preparation for the modelling phase. Ideally this module should provide a range of data manipulation facilities and some level of pre-modelling analysis so as to both service and focus the higher-order modelling processes.

Primary Modelling: A key requirement is the formulation of a model of a target time series. The simplest model is a univariate model based on historical records of the target series. ANTAS should therefore have the ability to hypothesise, test and validate univariate models of the target series.

Secondary Modelling: Once a primary model has been hypothesised there are two forms of secondary modelling that ANTAS should be capable of: firstly, the ability to extend the model to include data series that are predictive to the target series and therefore improve the overall model; secondly, the ability to analyse the residual performance of a model in an attempt to eliminate, or minimise, any obvious bias in the model's performance (for example, secondary conditions that correlate with the model performing well, or that signal model failure).

Model Integration: ANTAS must produce a predictive model of the target series. It is therefore imperative that the system has the ability to integrate primary and secondary modelling so as to produce a single coherent forecast of the target series (for example, the direction of a price trend).

System Monitoring and Configuration: A final level of functionality required within ANTAS is the ability to monitor its own performance. The system should have the ability to re-configure models on the basis of past performance.

In the context of the complete system we have the following procedure: hypothesise a univariate time series model based on the target series; validate and refine this model; then use this primary model as a benchmark to expand the modelling process to include secondary time series and secondary model analysis to form a *secondary integrated model.* Once the expanded model is in place a new round of validation and hypothesis testing can be conducted. Once this process is complete the model is ready for use.

Having specified the system's top-level requirements, the next few sections will expand these objectives into detailed sub-tasks.

6.2 ANTAS

Drawing on the results of the earlier chapters, and the functional requirements outlined in 6.1, we are now in a position to propose the full system design of ANTAS. We start by presenting a functional overview of the complete system and then throughout the remainder of the chapter specify the detailed modelling and control mechanisms that constitute its automated decision process.

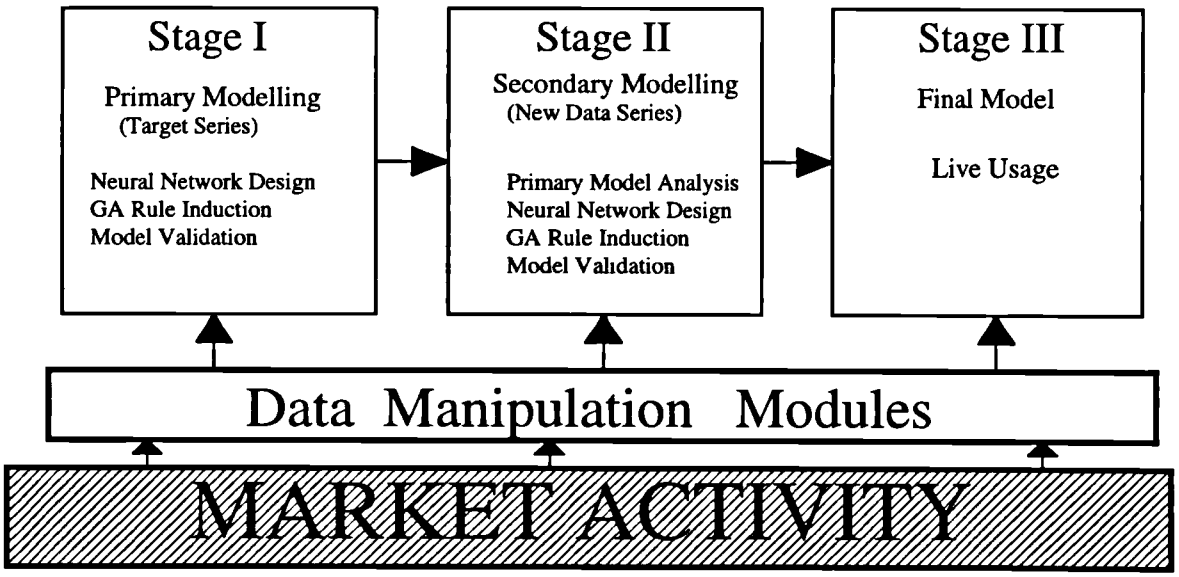


Figure 6.2.1: ANTAS - Functional Lay-Out.

Figure 6.2.1 presents the functional layout of ANTAS. The above can be read as an assembly line with time running from left to right. The purpose of the system is to hypothesise, design and produce a working model of a target financial series. The functional scope of the system can be broken down into three basic phases: Stage I, the construction and validation of primary models

(univariate analysis); Stage II, the construction and validation of secondary models (multivariate analysis); Stage III, the application and monitoring of the best model to emerge from stages I and II to the target domain. In Chapter 7 we simulate Stage III by a prolonged series of out-of-sample experiments. Whilst each of the separate stages are useful in discussing the system, it should be pointed out that modules that appear in Stage I are also used in Stage II, and that the stages more accurately represent phases in the modelling process. The specification for each of the modules is outlined below, with the functional layout given in Figure 6.2.2.

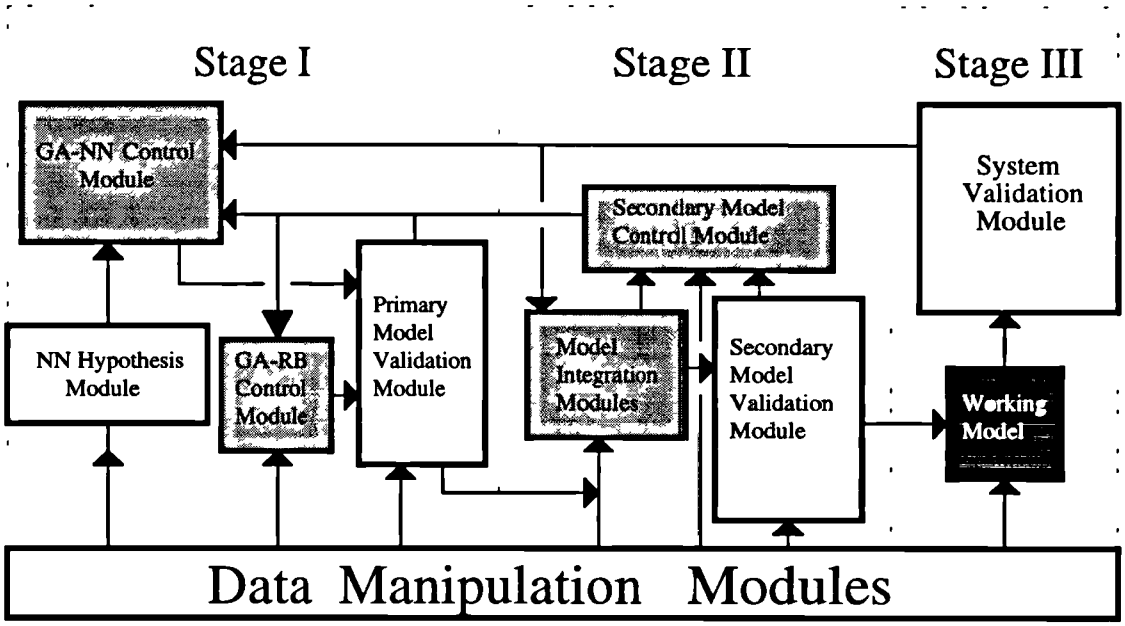


Figure 6.2.2: ANTAS - Modular Lay-Out.

Stage I: Primary Modelling

Market Activity: This refers to the data store for the whole system. Past price and volume series are stored along with available technical indicators. The precise data stored will be dependent on the target financial series that the system is to model. Data descriptions will be given in Chapter 7.

Data Manipulation Modules (DMMs): These modules house all the relevant data treatment routines. This includes linear, logarithmic and hyperbolic tangent scaling, along with facilities for producing moving averages. Other modules can call on any combination of the DMMs to effect a specified data manipulation task.

NN Hypothesis Module: This module comprises two sub-modules responsible for i) specifying the NN architecture, and ii) fixing network training parameters that relate directly to the data.

NN modules: These refer to single neural network models. Each is effectively a neural network slave with parameters and training conditions set via a GA control structure.

GA-NN Control Module: This module specifies the target series a neural network is trained on, instructs the neural network Hypothesis Module to fix parameters related to data and architecture,

and finally selects and fixes the learning parameters for a candidate neural network model by using a GA.

GA-RB Module: The GA Rule-Based (GA-RB) Module uses a GA directly to form predictive rules for a specified series. Rules can relate to the target series, or secondary data series, or on statistics collected from existing models. In most instances this module will be used as part of the integrated strategy required for secondary model formation.

Rule Models (RMs): These refer to single GA rules produced by the GA-RB module. Rules are induced as a means for both fine tuning existing neural network models, as well as a means of producing predictive models in their own right.

Primary Model Validation Module: This module conducts a v-fold validation of the models proposed by the GA Control Module and the GA-RB Module. Where necessary control is passed back to either control module if the model is judged to be defective. When a model is passed as satisfactory, this module is also responsible for collecting performance statistics for each of the neural network models in preparation for the secondary modelling phase.

Stage II: Secondary Modelling

Secondary Model Control Module: This module controls the choice of secondary data series that the system should model. Secondary series are chosen on the basis that they may be explanatory to the target series. Once this module finds a data set that meets fixed correlation requirements, it can request a primary model of the new data series from either the GA-neural network Control Module, or from the GA-RB Control Module.

Model Integration Modules: Models produced by Stage I consist of single neural networks (possibly trained with multiple time series) and rule-based models produced via the GA-RB Module. The purpose of Stage II is to integrate related models in an attempt to improve the overall predictive score for the target series. This module uses simple GA rules for combining primary models. The results of this phase have two consequences: firstly, multivariate neural network modules can be hypothesised and therefore control is passed back to the GA-NN Control Module; secondly, complete combined models can be passed to the Secondary Model Validation phase.

Secondary Model Validation Module: This is the final phase of validation and performance scoring. The systems model is validated via v-validation from which a full set of performance statistics are compiled. If the system's performance is poor then this module has the ability to either trigger a new round of model construction (Stage I) or model tuning (Stage II).

Stage III: System Modelling

Working Model: Having completed Stage II the system is now in possession of a satisfactory model. This is then ready for live usage. For this thesis this triggers the final round of performance measures. This set of results is treated as the final performance statistics and is the basis for the analysis of the system conducted in Chapter 8.

System Performance Module: The final module is dedicated to system performance. The task of this module is to analyse performance and where necessary trigger a new round of model construction and development based on live performance.

6.3 Primary Modelling

In the next few sections we provide a more detailed analysis of the sub-tasks involved in each of the modules described above, and present the details of the automated decision process within ANTAS. We start with the various methods employed for primary modelling.

6.3.1 Automating the use of Neural Nets

In terms of the functional specifications made in section 6.1 neural networks form the core of the modelling process and will be a vital part of the primary modelling phase. One of the most difficult tasks ANTAS must overcome is the automated construction of neural network models. As mentioned in 6.1, Table 3.1.1 of Chapter 3 presented a detailed list of parameters associated with the design of a neural network model. Drawing on the results of Chapters 3, 4 and 5 we now return to Table 3.1 and present the methods used in ANTAS to overcome the parameterisation issues, and where appropriate mark out the modules within ANTAS responsible for the design decisions.

DATA Selection: Data Manipulation Modules (Table 3.1.1&3.1.3)

The precise data requirements is problem specific and therefore will be covered in Chapter 7 when we discuss the experimental set-up and testing of ANTAS on a specified financial time series problem.

Specifying the Neural Network Model: Network Architecture (Table 3.1.4&3.1.6)

For each neural network a precise topology must be decided upon.

Approach (NN Hypothesis Modules):

In Chapter 5 Network Regression Pruning (NRP) was introduced. The method involves three stages. Firstly, the training of a network hypothesised to be overly large for the given learning problem. Secondly, the network is pruned according to NRP. Thirdly, the MSE in-sample error profile of the regression pruning curve is analysed according to the equations presented in Chapter 5 and a network complexity bound is hypothesised. The complexity bound C is then used in the network validation phase as a means for bounding the network topologies.

Forecast Horizon (Table 3.1.5)

In an automated system some means for fixing a target forecast horizon should be provided.

Approach (NN Hypothesis Modules):

The means for inferring an optimal forecast horizon may be dependent on the level of data pre-processing (for example if moving averages are involved). The precise mechanism used in ANTAS will be described in Chapter 7 when a specific target data series is introduced.

Activation Functions (Table 3.1.7)

Each unit in the network must have a specified activation function.

Approach (fixed):

ANTAS will use hyperbolic tangent activation functions at the hidden layer and linear output nodes. This is a standard backpropagation configuration. Evidence has been presented in the neural network literature [Ref95] that suggests that the choice of hyperbolic tangent is consistent with faster training times.

Learning Rule and Learning Style (Table 3.1.8&9)

The metric that will be applied during training for weight adjustment.

Approach (fixed):

Standard backpropagation with dynamic learning rates will be used by each of the neural network models. Batch training is also favoured to on-line training. The basis for both decisions is provided in Chapter 3 and relates to training times. Taking this approach we are still faced with two free parameters, the learning multiplier and the learning divisor (for adjusting the weight update rule). The precise levels for both of these parameters will be fixed by the GA-NN Control Module.

Convergence (Table 3.1.10)

The number of training cycles used for each neural network.

Approach (NN Hypothesis Modules):

A two stage approach is adopted; firstly a fixed in-sample MSE threshold is chosen for the network used in the NRP phase. The number of cycles required to attain the fixed threshold is recorded and then used for all subsequent networks inferred by the pruning procedure.

Cost function (Table 3.1.11)

The cost function for minimisation during network training.

Approach (fixed):

For network training ANTAS will use the MSE of the target and output of the network over the training set (as in standard backpropagation) [Chapter 3].

Generalisation (Table 3.1.12)

Network validation.

Approach (GA-NN Control Module):

ANTAS uses the DRGA search strategy as described in Chapter 4 on a small population of neural network models to fix the following set of parameters [Chapter 5]:

i) The learning rate multiplier (range [1.1,5.0]).

ii) The learning rate divider (range [1.1,5.0]).

iii) The network topology (bounded by a small contiguous range of fully connected topologies bounded by the complexity value provided by NRP).

iv) The scaling of the training data (range ± 0.5 and ± 0.9 to conform with the range of the hyperbolic tangent activation function range).

The GA is used to manipulate four chromosomes, one for each the parameter choices. The ranges available to each chromosome is fixed according to values given above. The learning rate multiplier and divider are fixed to the range [1.1,5.0] based on the observation that outside this range network training can result in oscillation and non-convergence [Chapter 3]. The scaling range is symmetric over the intervals [-0.5,0.5] to [-0.9,0.9] these values were chosen to cover the activation function range. Fitness evaluation was carried out according to the formula given in Chapter 5, namely:

$$Min\left[E = \frac{1}{v}\sum_{j=1}^{v}(E_{j_1} + E_{j_2})\right], \tag{6.1}$$

where v is fixed as the number of validation samples, and E_{j_1} is the MSE of the out-of-sample iterated forecast, and E_{j_2} is the MSE of the first difference of the iterated forecast and target values.

6.3.2. GA Rule Based Modelling

In 6.3.1 we have given details on how a GA can be used as a control module, fine tuning the parameters of a neural network model. However, it is also possible to use a GA as the basis for a modelling tool in its own right. For primary modelling, within ANTAS, this consists of a GA generating simple predictive rules for the likely direction of the target series over a forecast horizon based on past time series movement. The reason for including such models relates to the target domain of financial time series prediction. For finance an indication of the trend of future price movement is of extreme importance and forms the basis of many trading strategies. However, one of the main reasons for including price movement indicators within ANTAS is the fact that many authors have pointed out the extreme difficulty involved in forecasting price movement as compared to price direction for financial series [King95], therefore a more reasonable objective for ANTAS may be to predict price direction.

One of the simplest price movement rules that can be formed by the GA is of the following type:

$$R(t,k,T) = \text{Rule: IF } (P_t - P_{t-k} > T) \text{ then (R = 1) else (R = -1);} \tag{6.2}$$

where P_t is the value of the series at time t, P_{t-k} is the value at time t-k, and T is a threshold. The value of R corresponds to 1 for a price rise and -1 for a price fall. The GA's task is to find values for t, k, $0 \le t,k \le n$ (where n is the size of the *training* set) and T so as to maximise the probability of predicting a future price rise or fall over a fixed forecast horizon. In essence, the rule is a simplification of mean reversion theory, suggesting that large price rises or large price falls are subject to some form of market correction. If the data set is relatively small then a rule of the above form could be derived by direct search. ANTAS uses a GA to allow for extended

searches. Fitness evaluation is measured as the maximum hit value (correct prediction) for the rule over a large number of training sets. Explicitly we have:

$$Max\left[\frac{1}{N-n}\sum_{i=n}^{N} sign(P_{i+h}-P_i).R(t,k,T)\right], \qquad 6.3$$

where $sign(x)=1, x\geq 0, sign(x)=-1, x<0$, and h is the forecast horizon. Price movement models such as the one described above are carried out within ANTAS by the GA-RB module.

6.4 Secondary Modelling

The purpose of secondary modelling is twofold: firstly, to propose and construct more comprehensive models of the target process by including additional time series, and secondly, to investigate an existing model's performance in an attempt to infer conditions under which the model can be improved. This second objective attempts to find secondary conditions that when satisfied correlate with an improved performance of the model. Within ANTAS this will be confined to residual analysis of the neural network models. What is required is an automated means for detecting bias in the model's performance so as to infer conditions under which the algorithm can be expected to perform well, or equally conditions where its performance may be unreliable [GoFe94], [Pack90].

6.4.1 Generating Secondary Models

The Secondary Model Control Module is responsible for the selection of secondary data series considered influential on the target forecasting objective. For example, if ANTAS is used to forecast the Dollar-Pound (UK) exchange rate, then an often cited explanatory factor in exchange rate movements is the respective countries' interest rate levels [PeCr85]. It is therefore reasonable that a comprehensive exchange-rate model will include the sub-task of forecasting interest rates. To automate this form of expanded model construction the Secondary Model Control Module selects data series (via the Data Manipulation Modules) on the basis of high absolute correlation with the target series. Once a series that correlates is found, control is passed back to the GA-RM module which then constructs a rule based model of the new series. The new model can then be integrated with the existing target model via the Model Integration Modules.

6.4.2 Model Integration

There are several tasks associated with model integration, all of which are housed within what is termed the Module Integration Modules. The first task relates to inferring more complex models based on the performance of two primary models. As ANTAS is to be used for financial time series one of the main objectives of the system will be to forecast trends of the target series over a forecast horizon. ANTAS combines primary models of the target series by using GA rules based on the model's prediction for the forecast direction. The combined model is scored over data sets out-of-sample from those used in Stage I to produce the models. If the combined model shows a statistically significant improvement in performance (the number of times the correct trend is forecasted) then the new model is retained. If the combined model's performance is worse, then the best individual model is retained. If the combined model's performance is

approximately the same then all three models are analysed for bias (see below). If the primary models that are being combined relate to different primary series (for example a Dollar-Pound model and a UK-interest rates model as in the example above) then several actions are possible, but all start by a process of model integration. Model integration is achieved by using a GA to infer conditions under which a secondary series is predictive to the target series over a fixed forecast horizon. The process is similar to that presented in 6.3.2, with a slight change in the rules that the GA is used to infer (again it should be stressed this relates to price direction of the forecast horizon). Explicitly the rules induced are of the following form:

$R(t,k,T_1,T_2) = R$, where,

Rule: IF $(P_{i,t} - P_{i,t-k} > T_1)$ then (R = $dir(P_{i,h})$)

else: IF $(P_{i,t} - P_{i,t-k} < T_2)$ then (R = -$dir(P_{i,h})$)

where $P_{i,t}$ is the value of series i at time t, $P_{i,t-k}$ is the value at time t-k, and T_1, T_2 are thresholds. The value of R corresponds to 1 for a *target* financial series price rise and -1 for a price fall. In the above R is taken to be the direction, denoted *dir*, of the secondary series over the forecast horizon, $P_{i,h}$. The GA's task is to maximise the probability of predicting a future price rise or fall over the forecast horizon in the target series by selecting values for t,k,T_1,T_2. This form of rule induction is extended to more than one secondary series by the use of conjuncts in the rule formation (i.e., a conjunctive form for each of the states of R is induced). The fitness function used in all cases is simply the number of correct predictions over a number of fixed training sets.

Having produced a rule of the above form the next step is to combine the original models. This is done by using a GA rule and substituting the secondary model output for $dir(P_{i,h})$ in the above rule. The combined model is tested out-of-sample from data used to create the rule. If the model out-performs the original model by a statistically significant amount then a new combined neural network model is requested (via the Secondary Model Hypothesis Module) from the GA-NN Control Module. This will cause a new round of network design and testing using a neural network for multivariate analysis on the combined series.

If, however, the combined model's performance is worse than original model, ANTAS will try to form a rule that combines the new series and the target series directly (of the form given above). If the new combined model still performs poorly the secondary series is rejected. If not, a new combined neural network model is requested. In the cases where little change is seen in the combined model as compared to original model then ANTAS starts the process of model performance analysis.

6.4.3 Model Performance Statistics

An important aspect of the model integration phase is the analysis of an existing model's performance. Within ANTAS a standard set of performance statistics are generated by the Primary Model Validation Module and the Secondary Model Validation Module. The results for this thesis are principally concerned with the ability of the model to forecast the direction of the target series and are based on v-fold cross validation. The manner in which validation is

conducted is described in section 6.5 and is not directly related to secondary modelling, which is our concern here. What we are concerned with in this section is the information made available at the validation phase that can be used as the basis for extending an existing model and, for this thesis, this applies exclusively to neural network models.

Each neural network model is run as a slave process to either the GA-NN Control Module, or the Secondary Model Control Module. Each neural network training run generates a range of statistics relating to performance. At the end of neural network training, and as part of the *v*-fold validation process a Neural Network Multiple Run Analysis File is generated. This is depicted in Figure 6.4.1. for a univariate series.

Model NN-7	Target LGFC	Forecast 39 days	Moving Av: 34	Training Set:250			
Key	Target Direction	Raw Direction	Forecast Direction	Target Movement	Raw Movement	Forecast Movement	Hit
88	-1	-1	-1	-3.86	-2.62	-0.19	1
89	-1	-1	-1	-3.90	-2.22	-0.11	1
90	-1	-1	-1	-3.89	-2.86	-0.006	1
91	-1	-1	1	-3 91	-4.50	0.0001	0
92	-1	-1	-1	-3.95	-4.08	-0.014	1

Figure 6.4.1: Neural Network performance file.

The statistics and fields are interpreted as follows:

Model ID - NN-7: This is a unique identification of the model in question. Each model that is approved by the Primary Model Validation Module is given a unique ID, in this case NN-7.

Target - LGFC: This field gives the target series the model has been constructed for, in this case the LGFC (Long Gilt Futures Contract - see Chapter 7).

Forecast - 39 Days: This gives the forecast horizon for the model.

Moving Av: 34: If a moving average has been used, the number of days taken.

Training Set: 250: This gives the size of the training set for the neural network.

Key: This is a unique number associated with a validation experiment.

Target Direction: This field signals whether the target series experienced a rise or fall over the forecast horizon. A -1 denotes a fall, and a 1 denotes a rise. Note the target series in this case is the 34-day moving average of the LGFC.

Raw Direction: This field signals whether the raw series experienced a rise or fall over the forecast horizon. A -1 denotes a fall, and a 1 denotes a rise.

Forecast Direction: This field denotes the forecasted movement in the series.

Target Movement: The actual price movement experienced by the target series over the forecast period.

Raw Movement: The actual price movement experienced by the raw series over the forecast period.

Forecast Movement: The actual price movement forecasted by the model over the forecast period.

Hit: Indicates whether a correct forecast was made (1 indicates correct, 0 failure).

6.5 Validation Modules

Within ANTAS there are three validation modules related to various stages of the model production. The Primary Model Validation Module and the Secondary Model Validation Module both use v-fold cross validation on a section of data withheld from direct training. Both modules select v sections of data and test a model's performance. For financial series each model is scored on the number of times a correct forecast of direction is made. For this thesis v was varied depending on which stage of modelling was in progress [see Chapter 7].

Once the score for a model has been measured the result is treated as an estimate of the probability that the model will make a correct forecast of direction. This value is then compared to the probability of a series rise, or fall, as estimated from the v data sets (e.g., the probability of a price rise, or fall, over a fixed horizon based on the v data sets). If the model is performing badly then we should expect to see little difference between both statistics; conversely if the model is performing well we should expect a significant difference in both estimates, i.e., if we estimate the probability of a price rise, or fall, based on past price movement alone, we can estimate the probability of the model's score being due to random chance. In both validation models a significance threshold is set over which the model is accepted and below which the model is rejected. If the model is rejected control is passed back to the preceding stage within ANTAS.

The reason for using multiple selections of data for the v-fold testing is firstly to get a good estimate of the probability of direction based on past movement, and secondly so that we minimise the chances of the data being corrupted by over-training. Even with these provisos, for fair testing of the system the actual score of the model is not based on either of the statistics generated by the Primary Model Validation Module or the Secondary Model Validation Module – both of these results are judged to be *in-sample*. The model is scored via the Model Validation Module which for this thesis consists of exhaustive out-of-sample testing of the model hypothesised by the system. To ensure the validity of statistics generated by this module, all data for testing is used only once. That is, this module simulates live testing and once the system has been exposed to a data set a performance statistic is generated, and then the data is considered *in-sample*.

6.6 Control Flow

Having summarised the main modules, and described the means by which each module makes an individual contribution to model building, in this final section we describe the control structure within ANTAS. Each of the model building modules are for the most part autonomous, and require little to no knowledge of the activities taking place elsewhere within ANTAS. In order to co-ordinate and focus the system a series of control structures are responsible for

specifying the objectives for each of the various stages of the model: hypothesis, building, integration and validation.

6.6.1 Neural Net Control

The system makes use of four explicitly defined control structures (shaded grey in Figure 6.2.2). System control can either pass between these modules, or can be jointly shared between them. Each of the model building modules has a specific modelling task specified through a control file. For stage I the GA-NN Control Module runs multiple neural network slaves via a control file of the style depicted below:

Neural Net Control File:

```
#########################################################
#                    Control File 7
#          connects to: working7.table,  pdf7.pdf
#          Auto-generated Control File (J.Kingdon)
#          8/9/94      Tag: 7
#########################################################
#Topology
          1. Input: 15
          2. Hidden: 4
#Activation
          3. Hidden Layer(1=Tanh,2=Logistic,3=Linear): 1
          4. Output Layer: 3
#Patterns to use
          5. Total Patterns: 420
          6. Training Patterns: 200
          7. Justification (1=begining,0=end): 1
          8. Over-ride use all data (1=yes,0=no): 0
#Network Parameters
          9. Forecast period: 10
          10. Cycle limit: 1000
          11. Tolerance: 0.01
          12. Cauchy factor: 0.000000
          13. Momentum: 0.000000
#Data Pre-processing
          14. Moving average: 0
          15. Scaling: +0.8 , -0.8
          16. Normalized (1=set). 0
          17. Tanh (2=set): 0
          18. Batch(1=set): 1
#Training Parameters
          19. Learning Rate Max: 1 000000
          20. Learning Rate Min: 0 100000
          21. Learning Rate Multiplier: 1.100000
          22. Learning Rate Divider: 2 000000
#Directory (set by the Master file).
          23. Run_path:
          /cs/research/claudel/rodin/jason/vm/AutoTradSyst/Modules/
ANTAS
#Post Processing
          24. Build Regression Profiles: 1
#Multi Series and PDF parameters Analysis
          25  Number of Targets. 1
          26  PDF set  In  Field Target      Lags
```

```
27 Series_1:    2    2    15
28 Series 2:    3    3    0
29 Series_3:    5    5    0
30. Series_4:   7    7    0
#Control loops (over-rides Justification taking begining)
31 Start Pattern: 20
32 Multiple Tests: 1
33. End Pattern 450
34 Step 5
35 Job Finished 0
```

Figure 6.6.1: Neural Net control file.

Each control file is uniquely identified with each of the neural network slaves that the GA-Control Module is running. This allows for distributed processing if facilities are available, and also provides the GA control process with a means of identifying each neural network population member. Most of the parameters in the above are self-explanatory, ranging from topological specifications, training parameters (learning multipliers, activation function) through to post-processing facilities such as regression pruning. The control file can also be used to specify multiple training sessions either on the same data, or windowing through an extended series of data.

As an example, the control file above specifies the following experiment: a 15 input, 4 hidden node, 1 output node neural network to be trained on 200 patterns for a maximum of 1000 cycles, or a tolerance level of 0.01. The tolerance is the MSE of the in-sample neural network performance.

The network is also instructed to pre-treat the data from its raw format by scaling the *training set* to have values between +0.8 and -0.8. Training sets are specified outside of the control file by placing data in files that conform to a pre-defined raw data format.

The network is to use batch training, with hyperbolic tangent activation functions in the hidden layer, and linear activation at the output layer. Dynamic learning is also to be used with a maximum learning step of 1.0 and a minimum of 0.1. The multiplier for the learning rate is 1.1 and the divider is 2. A run path is specified which allows the GA control module to direct output from each of the neural networks to specified directories (running under UNIX). The neural network is also specified to build regression profiles after training. Multiple series and PDF files refer to more complex settings for the neural network set-up. The Pattern Descriptor File (PDF) allows for more detailed descriptions of data pre-treatment, or data reading. This facility applies for when the neural network is being used for pattern recognition tasks as opposed to time series analysis. For time series analysis the lag values (which make up the input layer) are specified for each of the series to be used for neural network training. In the case depicted above a single series is being used and therefore the lags values and the number of input nodes coincide.

The final control parameters (numbered 31 to 35) specify that training should start from pattern 20 and that a single neural network training session should be conducted. The network is then to move to pattern 25 (step value 5) and train on the next 200 patterns in an identical manner and again collect the results. It is to repeat this until it exhausts the full 450 patterns taking step 5.

6.6.2 GA Control

The GA control structure is depicted below:

GA Control File:

```
#######################################################
#                  Genetic Algorithm Control File
#                              13/12/94
#          ID: 1
#######################################################
#Controls
          1. Population Size: 10
          2. Number Generations: 200
          3. Number Chromosomes: 4
          4. Chromosome Range:
                    High: 255.00
                    Low: 0.00
#Biological Opertators
          5. Mutation: 0.020000
          6. Crossover: 0.600000
          7 Best Selected: 0.300000
#Alphabets
          8. Dynamic Representation (Set=1,Fixed=0): 1
          9. Number of Changes: 3
          10. Cycle 2,4,8
#Statistics
          11. Save Frequency: 50
#Neural Net Specifics
          12. Input: 6 - 20
          13. Hidden: 4 - 4
          14. Training Patterns: 250 - 250
          15. Moving Average: 5 - 49
          16. Muliplier: 1.100000 - 5.000000
          17. Divider: 1.100000 - 5.000000
          18. Scaling: 0.500000 - 0.900000
#Control
          19. Job Finished: 0
```

Figure 6.6.2: Genetic Algorithm Control File.

The first 4 control subsections relate entirely to the GA parameterisation, the first seven of which are standard GA controls. Best Selected refers to truncated selection and dictates the level of elitism used in each generation cycle. The Alphabets subsection allows for a dynamic representation GA (DRGA) as described in Chapter 4. Option 10 specifies the alphabet changes employed during a run. The Save Frequency fixes the rate at which full statistics from the GA are saved.

The Neural Net Specifics as named refer to the boundary specifications for the neural network parameterisation. For each of the neural network parameters controlled by the GA a range is specified in which the GA is to operate. This is then linked to a neural network control structure. For rule generation, a generic GA tool is specified as depicted above except for the neural network-specific requirements.

6.7 Summary

In this chapter we have outlined the design of the ANTAS system. We have shown how each of the system's sub-tasks are housed within separate modules and have described the functional relationship between them. Moreover, where appropriate we have made the distinction between system control parameters and parameters derived by the system's own calculations. In most cases we have attempted to allow the system full scope for configuring a model with the minimum of human intervention. In essence ANTAS provides a controlled structure for formulating neural network models for a target time series. The basic structure of the code is presented in Appendix B. In the next chapter we describe the financial data series used to test ANTAS, and describe the series of experiments used to test and validate the financial models produced by ANTAS.

Chapter 7

The Data: The Long Gilt Futures Contract

This Chapter describes the financial data series that is to be used to test ANTAS. In this chapter we introduce the Long Gilt Futures Contract (LGFC) and other trading series that will be used by ANTAS in Chapter 8 in order to formulate a predictive model of the LGFC. The Chapter provides a description of this data series in the context of financial forecasting, and in terms of the Efficient Market Hypothesis.

7.1 The Long Gilt Futures Contract

The Long Gilt Futures Contract (LGFC) is a financial instrument giving the option to buy or sell government debt at some time in the future. The size of a contract is a nominal £50,000 and the price is expressed in terms of a notional 9 percent long dated stock. The minimum step by which price can move is a 32nd (one tick) of one percent. The buyer of a contract is theoretically buying a nominal £50,000 pounds of gilt stock for delivery in the future, all return being capital gain on the instrument itself. If interest rates fall, the value of the contract is likely to rise because gilt-edged stock will rise in value. The initial margin for the LGFC is £500, with each tick representing a potential profit or loss of approximately £15 [Bret91]. Financial futures contracts very rarely result in physical delivery. The purchaser of a contract simply closes their position by selling an identical contract and taking a profit or loss.

The contract can be used in a number of ways. A trader might buy as a gamble based on the hunch that interest rates will fall. Alternatively, a buyer may have £50,000 to invest in gilt-edged stock in the future and may fear that prices will rise. From a forecasting point of view there are many factors associated with price movement in the series. Since gilt prices are often taken as indicators for the economy as a whole, a variety of different indices are thought to be relevant in fixing gilt prices [Bret91]. For example, statistics such as the Public Sector Borrowing Requirement (PSBR), the exchange rate, unemployment levels, and short- and long-term interest rates are all seen as contributory factors when pricing gilt edge stock. Gilt futures which are derivatives of gilt edge stock are therefore thought to be driven by speculation as to the likely strength or weakness of the future economy [PeCr85].

The above description of the factors that influence gilt prices makes the LGFC an extremely good candidate for almost all forms of the Efficient Market Hypothesis [Chapter 2]. The fact that the series is traded in discrete contracts also means that no single price series exists, and this must further reduce the possibility that future contract prices can be predicted from past contract price levels. From this perspective there would be little evidence to suggest that any internal dynamic within the price series exists, let alone that future price values could be forecasted.

From a conventional economic perspective there is therefore no reason to believe that the series is anything other than random. We shall test this hypothesis when assessing the actual data series and in terms of the performance of ANTAS [Chapter 8].

7.2 The LGFC Data

ANTAS is designed as a time series forecasting system and therefore the LGFC must be formulated as a time series problem in order to apply ANTAS. However, as mentioned previously, the LGFC is in reality a series of discrete contract prices which operate over a finite time interval. In this section we describe the necessary steps required in order to re-formulate the LGFC as a time series problem.

7.2.1 Time Series Construction

The fact that the LGFC is really a series of discrete prices relating to finite contracts brings us onto the first technical problem associated with forecasting the LGFC. A continuous series must be created from the various LGFC contract prices in order to construct a training set for ANTAS. Past contract prices can be obtained from LIFFE (London International Financial Futures Exchange). For each day that trading takes place the following information is recorded: opening range (high-low at market open), the volume of contracts traded that day and the closing high-low. This information is typically given for three contracts that are currently being traded. A typical example is given in Figure 7.2.1.

Trading Day	Tuesday 2 March 1993						
	Opening Range		Settle Change		Floor Daily		Est. Floor Vol.
Long Gilt							
Mar93	105-02	105-02	105-06	-0-05	105-08	105-00	4259
Jun93	106-02	105-31	106-05	-0-02	106-05	105-27	25899
Sep93	105-00	105-00	105-05	-0-02	105-00	105-00	50

Figure 7.2.1: LGFC price data from LIFFE.

The first column in 7.2.1 gives the end date for the contract under consideration. The Opening Range is the buy/sell price at the opening of the day's trading, and the Floor Daily gives the corresponding closing prices. The settle change refers to the price change from the mid-closing price of the previous day. The Est. Floor Vol. is the estimated trading volume for each of the contracts traded. One way in which to create a single series from the above is to concatenate the prices of given contracts over the time span of the full data series. A systematic way do this is to use the estimated floor volumes as indicators of the most traded contract, and to create a series consisting of the most traded contracts, that is, form a single series by choosing the opening low of the most traded contract for any given trading day. Using this method the price chosen for the 2nd of March 1993 depicted in Figure 7.2.1 would be 105-31, as this corresponds to the lowest opening price for the most traded contract (the June contract) for that day. The resultant price series for the LGFC is depicted below in Figure 7.2.2.

The technique described above is a common method used to create financial time series from discrete contracts, and is the method that will be employed in this thesis. Moreover, the series as depicted in Figure 7.2.2 is the way in which the contract prices are displayed in financial literature such as The Financial Times. For the complete series of experiments 10 years' worth of

daily LGFC data was made available from LIFFE. The series constructed consisted of all high volume contracts for the period 18/11/82 to 23/2/1993, giving 2596 trading days, with 124 changes of contract.

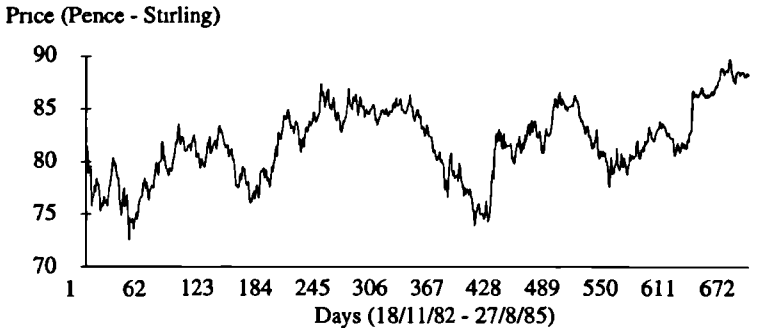


Figure 7.2.2: The LGFC constructed price series (18/11/82 - 27/8/85).

One factor that had to be taken into account for the global series was the fact that in the mid-1980s the way in which the LGFC price is calculated was changed. This is depicted in Figure 7.2.3.

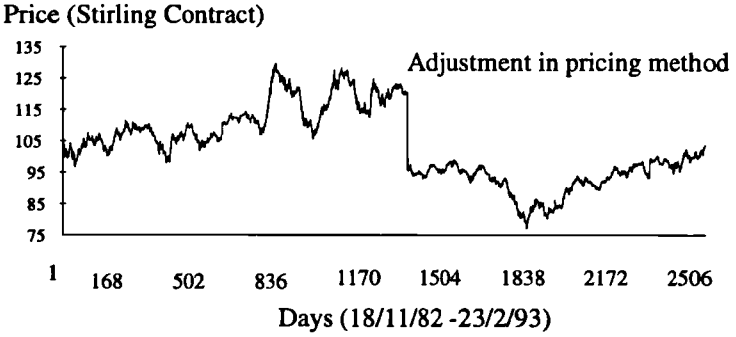


Figure 7.2.3: The LGFC Price series.

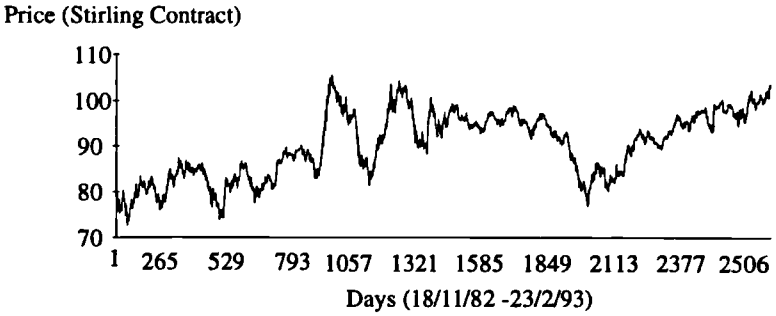


Figure 7.2.4: The LGFC adjusted price series.

Figure 7.2.3 shows there is a large price drop (24 ticks) in the contract series in the mid 80s. This was the result of a technical change in the way the LGFC contract instrument is priced by the Bank of England. To account for this the series up to this point was adjusted so as to reflect the current pricing method. This effectively meant that contracts before this date were offset by a price adjustment of 24 ticks. The adjusted price series is shown in Figure 7.2.4. The price series depicted in Figure 7.2.4 is the series that will be used in Chapter 8 for all LGFC modelling. In all

subsequent analysis the first 1000 data points of the LGFC series are treated as in-sample, and the remainder of the series is out-of-sample. The out-of-sample data is used to simulate live trading and is therefore not included in any analysis for designing the LGFC price trend model. This will be made explicit in Chapter 8. However, it should be mentioned that all analysis conducted in this Chapter only includes the first 1000 points of the LGFC data series.

7.3 Secondary Data

ANTAS is designed to take into account possible relationships between secondary data series and the target data series. For this thesis several secondary data series were available for ANTAS modelling. In traditional economic terms the additional series could be viewed as being influential on price movements within the LGFC, and therefore offer the possibility of improving the overall modelling capability for the target LGFC series. Four additional data sets made available to ANTAS were as follows:

LGFC Volume: The target LGFC price series is made from a series of discrete contracts. For each of the contracts the volume of trades was available for each day of trading. By concatenating this data a new time series is created based on the volume of the most traded contract. Correlation with the LGFC price series is 0.007. This value is remarkably low for what appear to be related events. If the absolute price movement in the LGFC is compared to the absolute movement in volume (i.e., to test the hypothesis that large volume corresponds to large price changes) this scores an even lower correlation coefficient (0.003).

Over 15 Year Stock Index: This a composite price index for all gilts traded that have redemption date longer than 15 years. The series is artificially created by the Financial Times using weighted values for the most traded gilts. This has correlation of 0.748 with the LGFC price series.

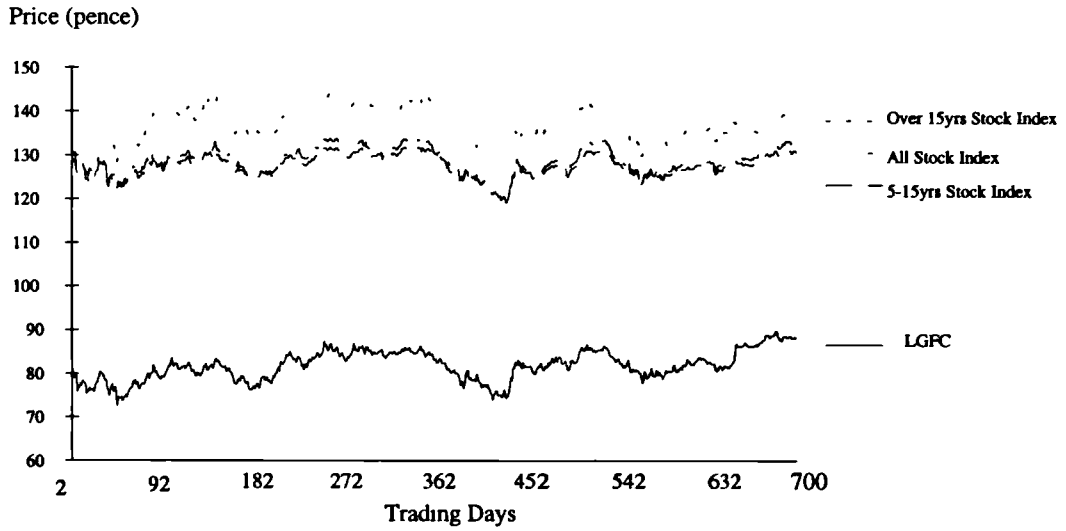


Figure 7.3.1: The Gilt Indices as compared to the LGFC for the training data (18-11-82 to 27-08-85).

All Stock Index: This a composite price index for all gilts traded. The series is artificially created using weighted values for the most traded gilts. This has correlation of 0.8520 with the LGFC price series.

5 to 15 Year Stock Index: This a composite price index for all gilts traded that have redemption date between 5 and 15 years. The series is artificially created using weighted values for the most traded gilts. This has correlation of 0.853 with the LGFC price series.

Figure 7.3.1 show the three gilt indices alongside the LGFC price series. Each of the gilt indices relate to gilts that are under active trading (as opposed to futures contracts). Each of the indices correlate extremely well with the LGFC and therefore may offer some explanatory aspects in terms of constructing a predictive LGFC model. One aspect that is disappointing in the above data is the poor linear relationship between the volume of LGFC contracts traded and the price. In a naive sense it seems reasonable to expect that price and volume should be closely related. These relationships will be explored more closely in Chapter 8 when we describe the Secondary Modelling Process within ANTAS.

7.4 Data Preparation

ANTAS provides a generalised methodology for modelling an arbitrary time series problem. Despite the general purpose nature of ANTAS there are specific data handling requirements that are unique to different time series problems. In this section we describe the necessary data pre-processing required for the LGFC, and the specific modules within ANTAS that are responsible for the data manipulation tasks. We also show how these modules integrate with the higher level modules described in Chapter 6.

7.4.1 LGFC Data Treatment

Having constructed a single price series for the LGFC several additional treatments are required in order to produce a data series suitable for Neural Network (NN) modelling. Firstly it should be recalled that the LGFC is priced in tick values, where one tick is 1/32 of a point movement. For example the opening price of the June contract given in Figure 7.2.1 is 105-31. This corresponds to £105.00 and 31/32 of £1, or 0.96875 pence, giving a price £105.96875. Each of the contract prices for the LGFC were adjusted in this way so as to produce a decimalised series. This series we refer to as the raw data series, or the raw target series.

Having done this the resultant time series can be used for neural network training. However, there are still likely to be problems with a neural network model of price movement due to the marked discontinuities between contract prices. That is to say, the LGFC price series is constructed from different contract prices; as the price moves from one contract to another there is likely to be large price jumps in the created series. One way to tackle this problem is to use a moving average of the raw series. If a centred moving average is taken then this implies that the forecast horizon of the price model must be extended to a point beyond the length of the moving average. Furthermore, if a moving average is used then the relationship between the moving average and the raw series will also have to be taken into consideration. The reason for this is that

movement in the moving average over short periods of time will be significantly different to the corresponding raw price movement over the same period. For example, the raw series could rise in value while the moving average, based on an overall trend, could decrease. The smoothed average of the series will always be easier to forecast, yet the real target is the movement of the raw series. The longer the time period over which the moving average and actual price movement are viewed, the closer the corresponding trends will be. This relationship is depicted in Figure 7.4.1. for a 34-day moving average of the raw data series.

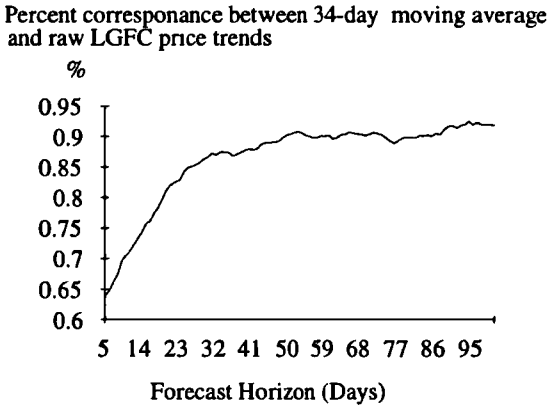


Figure 7.4.1.: The forecast accuracy for a 34-day moving average and raw LGFC price series.

What figure 7.4.1 shows is how the price trend in a 34-day moving average corresponds to trends in the raw series. It can be seen that at about 48 days the correspondence between the raw series and the moving average stands at about 88%. This means that a price rise or fall in a 34-day moving average over a 50-day period corresponds 88% of the time with a price rise or fall in the raw series over the same period. What Figure 7.4.1 provides is a means for deciding the forecast horizon that should be used for a given moving average. Generally speaking, we should prefer to forecast over the smallest possible forecast horizon. However, as Figure 7.4.1 shows, the shorter the forecast horizon for the moving average the less likely the results will have significance in predicting what happens in the raw series. Furthermore, on the basis of 7.4.1, a 50-day forecast horizon, for example, would represent the shortest forecast horizon for the 34-day moving average that has the greatest correspondence with raw series movement. For example, if we choose a 100 day forecast horizon the correspondence between trends is 92%, however for a 4% increase in the accuracy we have doubled the length of the forecast horizon. The 50-day horizon signifies the approximate turning point of the relationship, and affords the shortest forecast horizon with the highest likelihood of predicting the correct raw price trend.

7.4.2 Using Moving Averages

A further aspect of using a centred moving average is the fact that there is an inherent bias against the most recent price movements in the raw series. For example, for a 34-day moving average the last 17 days of raw price movement only contributes to one point of the moving average series, i.e., the last point in the moving average. This to some extent is a loss of information. It must be stressed that the movement and prediction of the moving average series is a surrogate for the real objective of forecasting the target raw series. The problem is best

illustrated with an example. Figure 7.4.2 shows a 34-day moving average of the LGFC along with the corresponding raw price movement. As we are taking a centred moving average, the moving average series available for training a neural network ends 17 days before the corresponding raw data series. This means a neural network will make its iterated forecast of the moving average series from a point 17 days before the end of the available raw data. This region is marked in grey in 7.4.2.

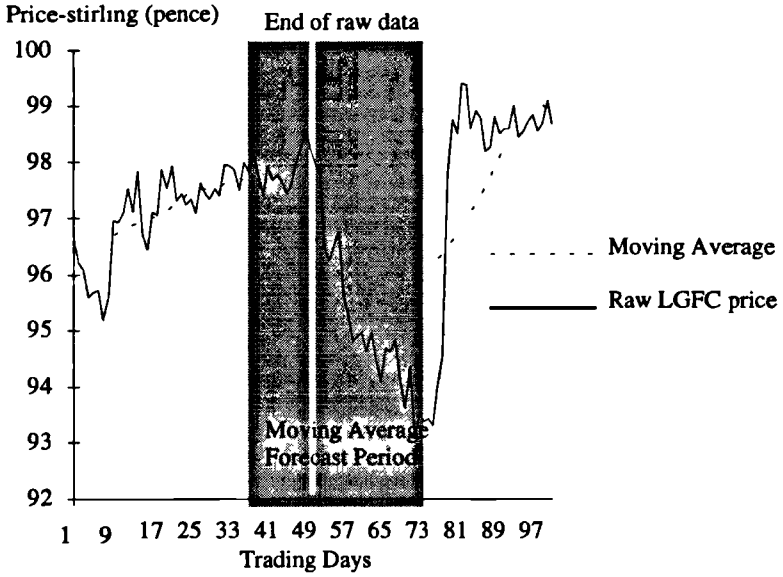


Figure 7.4.2: The moving average and raw LGFC series.

The white line represents the end of the raw data series (from the forecaster's view), and thus in a real-world example would coincide to the latest known price of the LGFC. As can be seen, before the forecast horizon the moving average has a rising trend, and thus a reasonable model could forecast a continuation of this trend and predict a price rise. However, if the last 17 days of raw LGFC price movement is taken into account, it can be seen that over that 17-day period a large price rise has already occurred which may signify that the forecasted price rise has already happened and in fact the series may be more prone to a fall, as indeed is the case in this example.

This style of analysis is an example of so-called Mean Reversion Theory (MRT) [Ridl94]. MRT suggests that a non-trending price series can be described in terms of an evenly placed probability distribution surrounding the real price of a stock. It postulates that movements into the tails of the probability distribution (as in large price rises or falls) has a tendency to be corrected with movement in the opposite direction (i.e., large price falls precede a *correcting* price rise).

MRT is a common chartist technique for financial analysis. It is regarded [Ridl94], [Refe95] as a useful method of trading for a non-trending price series. However, it should be pointed out that if it is known *a priori* that the series is non-trending then the technique is guaranteed to work in a tautological sense. In practise it is the skill of the trader to decide whether or not a trend is, or is not, occurring. To mimic this skill ANTAS will require methods for making this form of judgement, and will therefore also require means for combining this moving average prediction with other price series information in terms of formulating a model. To do this ANTAS will

record values for raw price movements that correspond to raw data movement in the period between the start of a moving average forecast and the last raw price value available for forecasting.

7.5 Data Treatment Modules

In order to carry out the various forms of data treatment described above ANTAS has facilities for linking with data treatment modules. ANTAS as far as possible is an automated system for time series analysis, we therefore must include ways in which automated data treatment is conducted for the LGFC. In particular we are interested in ways in which the moving average relationships described above can be carried out by the system in an automated fashion. In this section we describe these modules and provide explicit rules for formulating a forecast horizon for a corresponding moving average.

7.5.1 Moving Average Modules

Raw data treatment takes place within the Data Manipulation Modules (see Figure 6.1). An important aspect of LGFC modelling is the way in which moving averages can be used in relation to the raw price movement of the series. Within ANTAS this also has implications for the neural network Hypothesis Module. The NN Hypothesis Module's task is to formulate a neural network design and to set training parameters. For example, at some stage a target forecast horizon must be chosen. Within ANTAS this is achieved by a combination of the Data Manipulation Modules (specifically moving average generation) and the NN Hypothesis Module. The objective is to find a forecast horizon that maximises the likelihood of a correspondence between a trend in the moving average and a trend in the raw price movement. To do this a threshold was placed on the length of forecast horizon corresponding to a given moving average. This used the following relationship: Moving average forecast horizon threshold = H, where H is given for moving average K by:

$$S(h,k) = \sum_{t=1}^{T} sign(a_{k,t} - a_{k,t+h}).sign(x_t - x_{t+h}), \tag{7.1}$$

$$H = h \text{ such that } \left[\frac{S(h,K)}{T}.100\right] > 88\% \text{ for } 1 \le h \le 100, \tag{7.2}$$

where $a_{k,t}$ is the centred value for a K moving average for the price series $x_1, x_2, \cdots, x_t, \cdots x_T$. T is the size of the training set, h is the forecast horizon taken over values $1 \le h \le 100$, and $S(h,K)$ is the number of times a given forecast horizon price movement in the moving average corresponds to a raw price movement (in terms of direction).

For each forecast horizon we calculate $sign(a_{k,t} - a_{k,t+h}) = 1$ if $a_{k,t} - a_{k,t+h} \ge 0$ and $= 0$ otherwise. The summation is therefore the number of times that the raw price movement and the moving average price coincide in terms of direction over a time period h.

The result of expression 7.1 and 7.2 is a graph similar to that shown in Figure 7.2.3. What we are interested in is a forecast horizon h that has *at least* an 88% correspondence with the raw series

movement. Table 7.1 depicts the thresholds for some of the results for the moving averages for the LGFC and the full results are graphed in Figure 7.5.1.

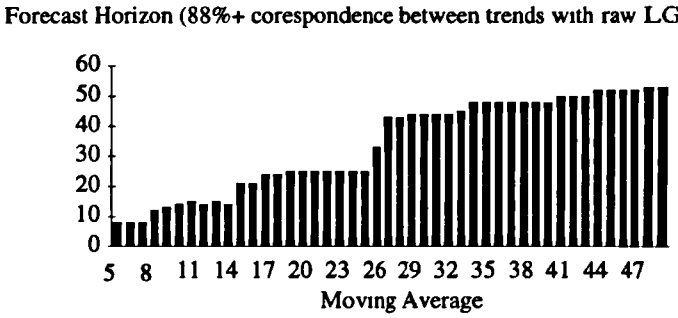


Figure 7.5.1: LGFC moving average and raw series price movements (18/11/82 to 27/8/85).

Moving Average	Horizon	Score	Moving Average	Horizon	Score
5	8	0.91	30	44	0 88
6	8	0.89	31	44	0.88
9	13	0.89	34	48	0.88
10	14	0.88	35	48	0.88
20	25	0 89	40	48	0.88
21	25	0.89	41	50	0 88
24	25	0.88	44	52	0.89
25	25	0.88	45	52	0.88

Table 7.5.1: Moving Averages Forecast Horizons for the LGFC (18/11/82 to 27/8/85).

The above provides a means for choosing a forecast horizon on the basis of a moving average in an automated fashion. These relationships are used by the GA-NN Control module as part of the full neural network optimisation process [Chapter 6 - 6.3.1]. Table 7.5.1 is created from in-sample data for the whole system.

7.6 Efficient Market Hypothesis and the LGFC

In this final section we conduct some exploratory analysis of the LGFC price series in order to determine the likely difficulty in forecasting this series. In particular we have mentioned the fact that in terms of traditional economic analysis there is no reason to believe that the LGFC is not included within the Efficient Market Hypothesis and that as such the series should therefore be random.

There are no rigorous tests that can be applied to the series in order to determine whether it is in fact random [MeSW81]. However, it is possible to look at the probability distribution of price movements so as to gauge the likely difficulty involved in terms of forecasting price trends. Figure 7.6.1 gives the histogram of daily price movement for LGFC for the first 1000 points of the data series. As can been seen, price movement is approximately normally distributed with the central bar (representing price changes of between -0.1 and +0.1 movement) scoring the highest, with a symmetric distribution of price rises and falls either side of the mean. The mean price movement is given by 0.0044 pounds (Stirling) over the 1000 days.

For the 1000 days the probability of a price rise was given as 0.51, and a corresponding 0.49 probability of a price fall. Testing the hypothesis that the probability of a price rise is in fact 0.5,

i.e., an even probability of a price movement which would be consistent with the series being random as predicted by the Efficient Market Hypothesis we have:

Null Hypothesis: $p = 0.5 = p_0$.

Alternative Hypothesis: $p \neq 0.5$.

Using the standard z statistic we can test the likelihood of a score of 0.51 versus 0.5 over the 1000-day price series. Taking estimates for the mean and standard deviation we have,

$$z = \frac{0.5 - \bar{p}}{\bar{\sigma}_{\bar{p}}}, \text{ where } \bar{\sigma}_{\bar{p}} = \sqrt{\frac{\bar{p}.\bar{q}}{N}}, \text{ with } \bar{q} = 1 - \bar{p}, \text{ and } N = 1000. \qquad 7.3$$

For this we get a z statistic of -0.63, which is well within a single standard deviation and therefore reasonable evidence that the price movement has no probabilistic bias to rise or fall over a given day. This is therefore consistent with the Efficient Market Hypothesis.

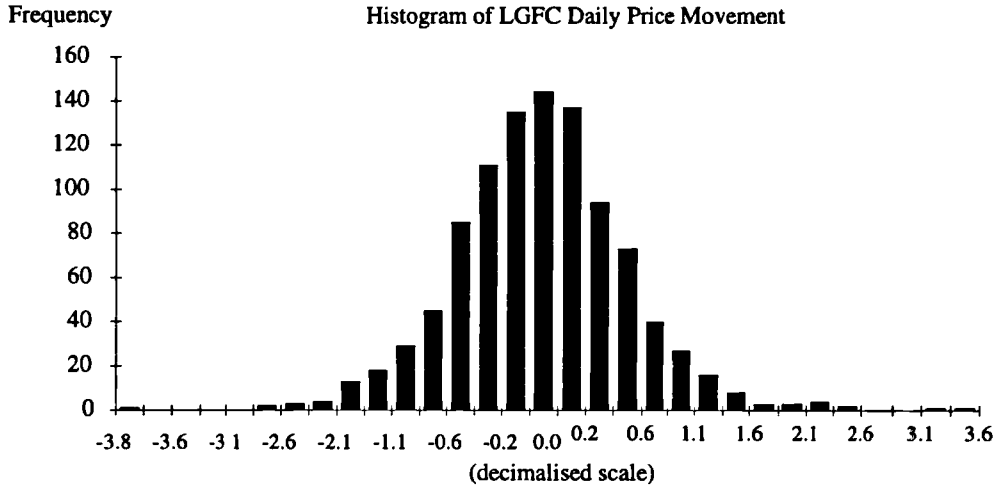


Figure 7.6.1: Histogram of LGFC daily price movement (18/11/82 to 27 8/85).

7.7 Summary

In this Chapter we introduced the data series that is to be used to test ANTAS. We have described the way in which a price time series can be constructed for the LGFC contract prices and have introduced related gilt series that may be beneficial in terms of forecasting the LGFC. This chapter has also described the way in which moving averages, and data manipulation modules specific to the LGFC, are included within ANTAS. Finally, we have provided some background analysis of the LGFC price series and shown that the series' daily price movement is consistent with the series being *efficient*, and therefore random. In the next chapter we describe the full series of experiments used to test ANTAS's ability to formulate price trend models of the LGFC price series.

Chapter 8

Experimental Results

This chapter provides a step-by-step critical analysis of the ANTAS system as applied to the Long Gilt Future Contract (LGFC) series. It is shown how both Primary and Secondary models are constructed in order to formulate a predictive model of price trends in the LGFC. Once the model is formulated 1000 out-of-sample experiments are conducted in which ANTAS is tested under "live" trading conditions. It is shown that the Neural Network (NN) model designed by ANTAS scores 60.6% in terms of a correct forecast for price trends in the LGFC. The results are then analysed in terms of the Efficient Market Hypothesis.

8.1 Experimental Design

In order to test ANTAS a comprehensive series of experiments had to be designed. The target for the system was the LGFC constructed price series for which ten years' worth of data was available. The series comprises all high volume contracts for the period 18/11/82 to 23/2/1993, giving 2596 trading days, with 124 changes of contract. This chapter can be divided into three stages of model development and analysis. In the first stage (Phase I) we show how ANTAS is used to construct a Primary Neural Network model for the univariate time series constructed for the LGFC. Once this model is constructed in Phase II we show how the GA Rule Based Module is used to infer a Secondary model that includes data other than the LGFC price series (i.e., makes use of LGFC traded volumes, the All Gilt Index, the 5-15 Years Gilt Index and the Under 5 Year Gilt Index). Once a secondary model has been constructed this is then compared with the Primary model and a decision is made as to which model will constitute the ANTAS system model to be used for out-of-sample testing. In Phase III we present the out-of-sample testing of the LGFC ANTAS model. This will consist of 1000 experiments in which the model is used to forecast future price trends of the LGFC as if under live trading conditions. In order to simulate live trading conditions the ANTAS model will be run once through this data with all parameters held fixed. That is to say, the model will not be adjusted (irrespective of the results) once this phase of testing is under way.

In terms of partitioning the data, for each stage of the testing the first 1000 data points of the LGFC series (and secondary series) were used for model construction (Phases I and II) and the remaining data was used for out-of-sample testing. In all tests and analysis conducted in Phases I and II the out-of-sample data was not used, and was kept strictly isolated from all forms of analysis. In this way the results achieved in Phase III can be regarded as a true reflection on how the system would have performed under live conditions. Phase I, II and III correspond to stages I,II and III given in Figure 6.1 [Chapter 6].

8.2 Phase I - Primary Models

To test ANTAS effectively the various modules within the system have to be assessed individually. The reason for this is that the decision process within the system is hierarchical and

therefore we would like to see a steady improvement in the system's forecasting capabilities as the models are combined. At each stage of the model construction process various statistical tests are conducted. The statistics generated at each stage are used for two purposes: firstly as control parameters for each of the control modules within the system for secondary modelling, and secondly as performance measures in their own right. The primary models that are of most interest for system analysis will be the neural network models constructed during Phase I. The reason for this is that they present the results that would have been achieved if a single neural network model was used in isolation without the added analysis provided by the remainder of the system.

8.2.1 NN Hypothesis Modules (Phase I)

The first stage in primary model construction was the hypothesis phase for a neural network trained on the LGFC. A 20-10-1 MLP was trained on the LGFC price series using the network training parameters detailed in Chapter 6. This consisted of 250 trading days of data using batch weight update, hyperbolic tangent activation functions for the hidden layer, linear output node, a data scaling of ± 0.8, a learning multiplier of 1.1 and divider of 2.0. The network was trained until the in-sample error fell below a Mean Square Error (MSE) of 0.01 (approximately 750 batch weight updates). This network then formed the basis for the Network Regression Pruning (NRP) phase of the network hypothesis module. The reason 250 trading days was chosen was that this corresponded to approximately one years' worth of trading data. The network regression pruning error profile is depicted below.

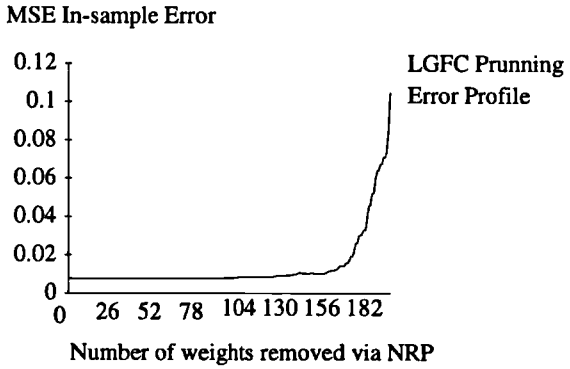


Figure 8.2.1: NRP applied to a 20-10-1 network trained on 250 days of the LGFC.

Topology	Complexity (weights)	Dist. From Target (78 8)
20-3-1	67	-11.8
19-3-1	64	-14.8
18-3-1	61	-17 8
17-4-1	77	-1.8
16-4-1	73	-5 8
15-4-1	69	-9.8
14-4-1	65	-13.8
13-5-1	76	-2.8

Topology	Complexity (weights)	Dist. From Target (78 8)
12-5-1	71	-7.8
11-6-1	79	0.2
10-6-1	73	-5 8
9-7-1	78	-0 8
8-7-1	71	-7.8
7-8-1	73	-5 8
6-9-1	73	-5 8
Out of Range		

Table 8.2.1: The hypothesised complexities for a MLPs used for the LGFC.

Using Equation 5.8 [Chapter 5] for the pruning error curve given in Figure 8.2.1, the complexity for a Multi-Layer Perceptron (MLP) trained on the LGFC was hypothesised as being 78.8. This value is translated into 15 candidate MLP architectures. These are given in Table 8.2.1. Recall the complexity refers to the number of weights in a network, which for totally connected 3-layer architectures provides the equation $complexity = In \times Hidden + 2 \times Hidden + 1$. Where *In* is the number of input nodes, *Hidden* the number of hidden nodes. The form of the equation also takes into account the bias nodes (1 bias for each hidden node and one for the output node).

Having restricted the number of candidate architectures the next phase in the neural network hypothesis procedure was to use the GA-NN Module to fine tune all training parameters and to fix the final choice of network topology. This includes the exact moving average and forecast horizon. For the 1000 days of in-sample data the GA used Table 7.5.1 to fix lower thresholds of possible forecast horizons for each moving average. Recall that the GAs task is to optimise the scaling of the LGFC data, the moving average of the data, the forecast horizon and the learning multiplier and divisor. For each of the neural networks in the GA population 250 days' worth of training data was used to train each network. The exact portion of data was chosen randomly from the in-sample data.

8.2.2 Results for GA-NN Module

The GA configuration was not fixed, with the exact parameterisation being left as a control parameter. This proved problematic as the GA did not produce good convergence profiles for neural networks that were trained on multiple randomly selected training sets (as advocated in Chapter 5). The training times required for each of the neural networks in the population also proved prohibitive. The GA was therefore restricted in the cases of multiple evaluations to a small population of 10 neural networks, and 50 generations. Both of these figures are small, and ideally a much larger population and larger number of generations would be used. However the most difficulty arose from the convergence characteristics of the GA.

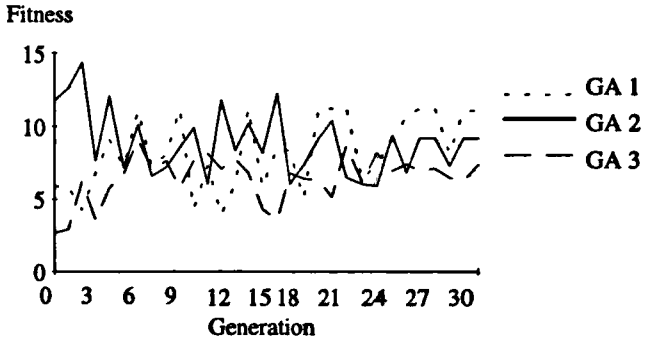


Figure 8.2.2: Three typical GAs applied to multi-evaluation time series fitness.

If each of the neural networks in the population were restricted to a single training set then good convergence could be achieved with a mutation rate of $1/L$ where L is the length of the binary string, a 0.65 rate of crossover, a 0.4 truncated selection and three base changes between 2,4 and 16. This presented a dilemma in terms of how the GA should be used. If the neural networks are optimised for a single training set then over-training would appear more likely. To

test which procedure produced the best results several GAs were run with different GA parameter settings.

Figure 8.2.2 shows a typical fitness profile for three of the GA's runs over 30 generations using multiple training sets. As can be seen, in each case the GA fails to converge to a single high fitness value. There are several reasons why this occurred. Firstly, neural networks that are trained on different training sets can legitimately produce varying fitness values. For example, if the GA isolates a good candidate neural network that uses a certain configuration for training, and the net scores well over a fixed training set, then within ANTAS a variation from this net produced by the GA may well be trained over a different training set (i.e., the training sets are selected randomly). The reason for using randomly selected training sets was to encourage good generalisation. However, because of the prohibitive running times, nets could only be trained over a small number of training sets (the maximum that was used was 2). For this small number of sets, the GA seemed unable to find networks that could score well over both training sets, *and* that were robust enough to score well if some small variations in the training parameters were introduced. This relates to the sensitivity of neural network modelling described in Chapter 3.

In order to evaluate this problem single training set neural networks were also tested. Adopting this approach produced much higher fitness scores and good convergence profiles for the GA. However, the danger of over-fitting the training set was also much more likely by reducing the number of tests that each neural network was subjected to. To illustrate this Figure 8.2.3 is the neural network configuration file produced by the best neural network found by the GA on one of the reduced training set runs. Figure 8.2.3 gives each of the parameters that were found by the GA optimisation process.

GA Produced NN Configuration - NN1	
Architecture	15-4-1
Forecast Horizon	50 Steps
Moving Average	23 Days
Scaling	0.58
Learning Multiplier	2.23
Learning Divider	1.1

Table 8.2.2: Neural Network Configuration produced via the GA.

NN-1 scored highly compared to the neural networks produced via GAs given in Table 8.2.2. The iterated forecast for this network is given in Figure 8.2.3.

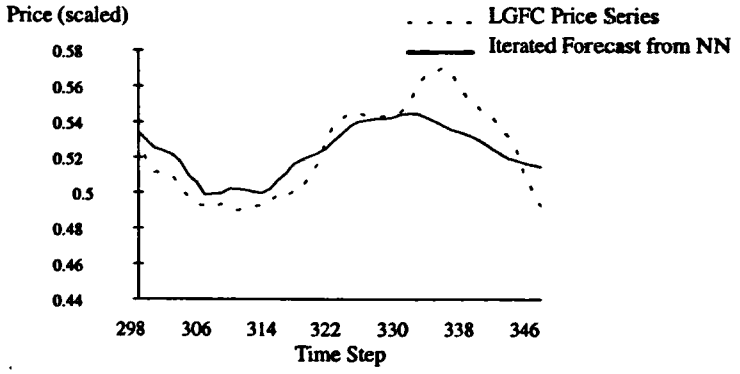


Figure 8.2.3: Iterated forecast from NN-1 produced by GA (15-4-1 Net).

As can be seen, the iterated forecast is extremely good, with the network producing an output that correctly forecasts two turning points over the 50-day period in the LGFC price series. However, a demonstration of how this result is essentially an over-training of the network is given in Figure 8.2.4. This shows the same network's response to an out-of-sample training run. In Figure 8.2.4 the net has produced a square wave approximation to the in-sample training data. This is due to the high learning rates that were found by the GA. The rates, given in 8.2.2, are essentially too finely tuned to the original training sets used by the GA. To confirm this Figure 8.2.5 gives the in-sample response of the same network as trained by the GA.

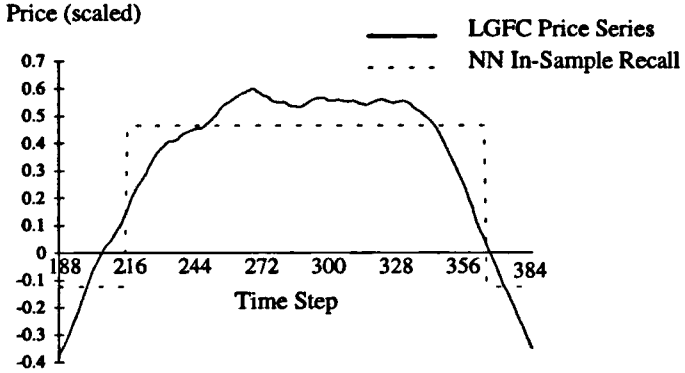


Figure 8.2.4: Corrupted In-Sample Recall from NN-1 (caused by too large learning rate).

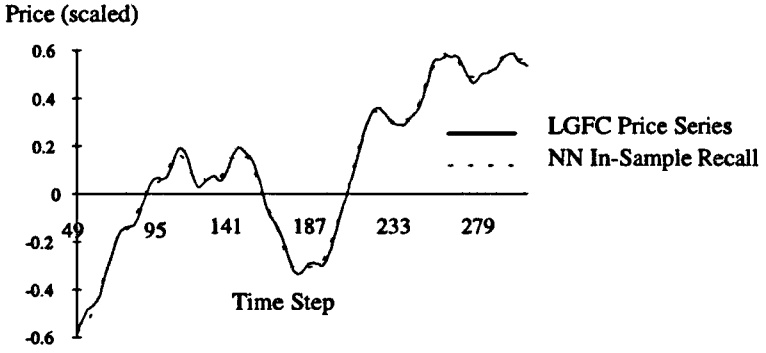


Figure 8.2.5: In-Sample Recall for NN-1 for GA-In-Sample training set.

Figure 8.2.5 gives a clear indication that for the portion of data used by the GA, NN-1 produces an extremely good in-sample approximation of the training data. This suggested that the initial range for the learning multipliers should be decreased, or that the learning divisor should be forced to be greater than the learning multiplier.

What Figures 8.2.3, 8.2.4 and 8.2.5 demonstrate is the real possibility of over-training if the GA is left to optimise the neural network parameters over a single training set. However, what Figure 8.2.2 demonstrates is the fact that if multiple training sets are used in the GA optimisation process then good GA convergence seems unlikely if only two samples of data can be used. In order to solve this problem single training sets were kept, but a restriction was made on the learning rates for each of the neural networks produced by the GA. In order to test for generalisation the best neural networks produced by several GA runs were tested on multiple data sets that were out-of-sample from the GA runs. The three best neural networks produced by the GA-NN module were as follows:

GA Based NN Configuration - NN-2		GA Produced NN Configuration - NN-3		GA Produced NN Configuration NN-4	
Architecture	15-4-1	Architecture	11-5-1	Architecture	12-5-1
Forecast Horizon	50 Steps	Forecast Horizon	50 Steps	Forecast Horizon	25 Steps
Moving Average	23 Days	Moving Average	23 Days	Moving Average	20 Days
Scaling	0 5	Scaling	0.44	Scaling	0 48
Learning Multiplier	1.1	Learning Multiplier	1.9	Learning Multiplier	1.29
Learning Divider	2 0	Learning Divider	2.85	Learning Divider	1 71

Table 8.2.3: Neural Net configurations inferred by GA design process.

The networks NN-3 and NN-4 were produced by the GA with a restricted rate of learning multiplier (the range [1.1,2.9]). NN-2 is a variation of NN-1. The architecture of NN-1 is retained along with most of the training parameters. However, the learning rates have been reduced and the scaling of the data was placed at 0.5. All three nets were then tested on 200 new data sets. Over the 200 experiments, and for the corresponding forecast horizons the LGFC price series had probability of 0.52 of a price rise, and 0.48 of a price fall for a 50-step forecast horizon. For a 25-step forecast horizon the probability was given as 0.445 for a price rise, and 0.555 for a price fall. This suggested that the 12-5-1 net may have an advantage over this period as there is a bias towards a price fall for the period, i.e., if the network simply predicted a price fall over the whole period a score of 55% would be attainable. The results are given below in Table 8.2.4.

Neural Net	Architecture	Forecast Horizon	% of Correct Trend Predictions for 200 LGFC Experiments	% of Price Rises for 200-day Period
NN-2	15-4-1	50	73.5%	52%
NN-3	12-5-1	25	69%	44.5%
NN-4	11-5-1	50	68.5%	52%

Table 8.2.4: Neural Networks tested over 200 day contiguous LGFC price series.

As can be seen, NN-2, with architecture 15-4-1, does the best of all the networks, despite the fact that there is significantly less predictability (in terms of an overall trend) associated with a 50 step forecast, as opposed to the 25-step forecast. Moreover, we can test the probability of a correct forecast being 0.73, compared with random chance of 0.52 with the following: let $\overline{p} = 0.52$, and take test statistic z,

$$z = \frac{0.73 - \overline{p}}{\sigma_{\overline{p}}}, \text{ where } \sigma_{\overline{p}} = \sqrt{\frac{\overline{p}.\overline{q}}{N}}, \text{ with } \overline{q} = 1 - \overline{p}, \text{ and } N=200, \qquad 8.1$$

then the probability of scoring 0.73 by chance gives a z statistic of 5.9445, which is less than 0.001, and therefore a statistical confidence level at over the 99% level. On this basis NN-2 was chosen as the candidate network for the LGFC primary data series.

The next step was to establish the performance of NN-2 over a larger portion of the in-sample data so as to form a basis upon which secondary modelling could start.

8.2.3 In-Sample Testing and Validation of the 15-4 Neural Network

The 15-4-1 network, NN-2, was tested over 500 days of the LGFC price series, which corresponded to the period 18/11/82 to 27/8/85. Over this period the network scored 60.4% correct prediction. As was the case for the 200 days analysis it is important to establish how well

the network is doing as compared to random chance. For the 500 days experiments the probability of a price rise was given as 0.478, and a corresponding 0.522 probability of a price fall. This represents almost a complete reversal of the statistics for the same series over the 200 days period. It is therefore worth testing the hypothesis that the probability of a price rise is in fact 0.5, i.e., an even probability of a price movement which would be consistent with the series being random as predicted by the Efficient Market Hypothesis. We therefore test:

Null Hypothesis: $p = 0.5 = p_0$.

Alternative Hypothesis: $p \neq 0.5$.

Using the standard z statistic we can test the likelihood of a score of 0.522 versus 0.5 over the 500 experiments. For this we get a z statistic of 0.98, which is well within a single standard deviation and therefore a reasonable working statistic. Moreover on this basis a score of 60.4% correct forecast direction for the neural network is significant at well over 99%. This is an extremely encouraging statistic, suggesting that some level of forecast success is available for this series. However, it must be recalled that the forecast horizon has been calculated on the basis of the full 500 data points, and therefore strictly speaking, the network is still operating in-sample. It will only be after the full out-of-sample tests are conducted that any firm conclusions can be offered. What these results do offer is the basis on which secondary modelling can start, in terms of both GA analysis of the model's behaviour, and the introduction of secondary series. We start with the GA-modelling phase.

8.3 GA-RB Module and Combined Validation

The GA-RB module provides the second means for forming primary models, i.e., models that only use information available from the target series. This can be achieved in two ways, firstly as GA rule based models of the price series, and secondly as analysis rules for correcting the neural network primary models. For this thesis, only this second type of analysis will be conducted. This decision is based solely on the amount of time and resource available for experimentation, and further work should include GA-RB models that are independent of the neural network models. For ANTAS the first phase in constructing GA-RB models starts with a full analysis of the performance of the neural network primary models. This implies that a deeper understanding of the candidate network NN-2 is the first step in the application of the GAs to the modelling process.

To start this analysis, we first get a more detailed analysis of how NN-2 achieved the score of 60.4% over the in-sample data.

Figure 8.3.1 shows the histogram of correctly forecasted price movement over the 500 days' worth of experiments. The X-axis corresponds to price movements of 0.2 decimalised tick movements in the LGFC. That is, price movement of approximately 6 ticks make up the frequency bands. As can be seen, the histogram appears to be evenly distributed with the price movement concentrated around zero.

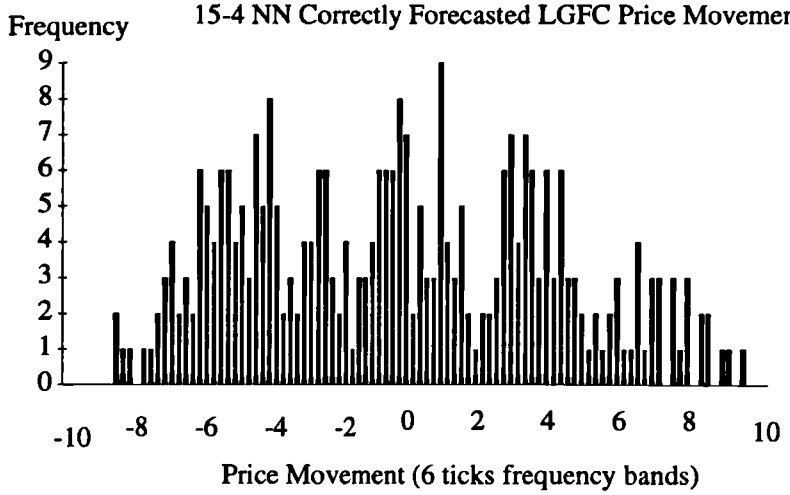


Figure 8.3.1: Histogram of correctly forecasted price direction for the LGFC (NN-2 for the period 18/11/82 to 27/8/85).

Figure 8.3.1 shows that NN-2 performs over a range of LGFC price movements. To check this observation, Figure 8.3.2 shows incorrectly forecasted price movement for NN-2.

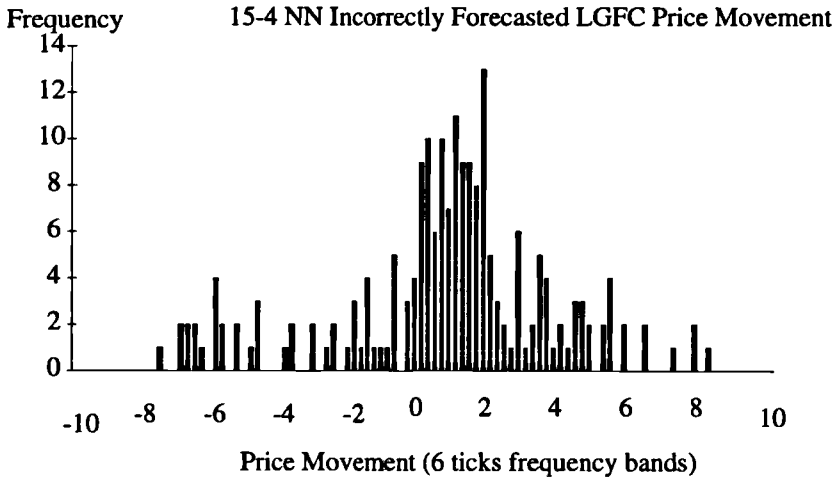


Figure 8.3.2: Histogram of incorrectly forecasted price direction for the LGFC (NN-2 for the period 18/11/82 to 27/8/85).

In Figure 8.3.2 we get some idea that the network performance may be slightly biased into forecasting a price fall as opposed to a price rise. Much of the error for NN-2 seems to accumulate around small price rises in the LGFC.

To get a clearer idea of any bias within the forecast we take the histogram of the forecasted correct price direction minus the incorrectly forecasted direction. This is depicted in Figure 8.3.3. Figure 8.3.3 provides a good way in assessing the quality of NN-2 forecasts in relationship to the price movement of the LGFC. The negative bars indicate where the network is has incorrectly forecasted LGFC price direction.

In Figure 8.3.4 we can affirm the observation that NN-2 does badly over the range of LGFC price change of between 0 and £2. Figure 8.3.4 provides the probability histogram of correct price prediction of the LGFC for NN-2.

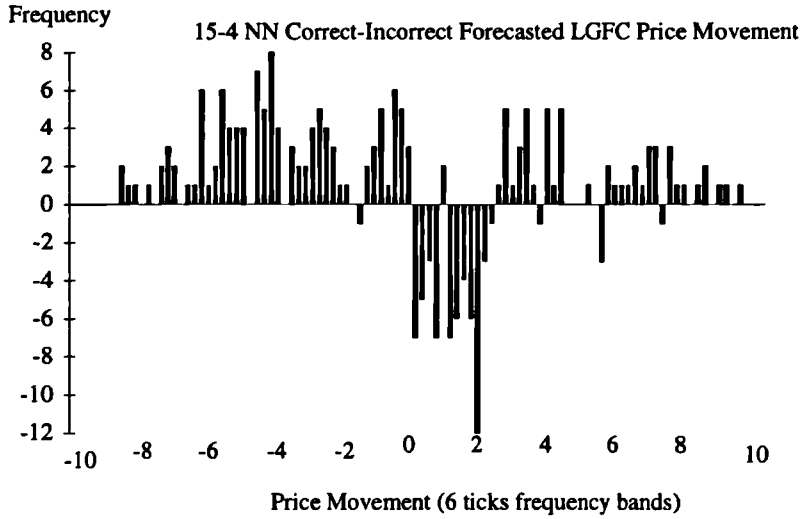


Figure 8.3.3: Histogram of correct minus incorrect forecasted price direction for the LGFC (NN-2 for the period 18/11/82 to 27/8/85).

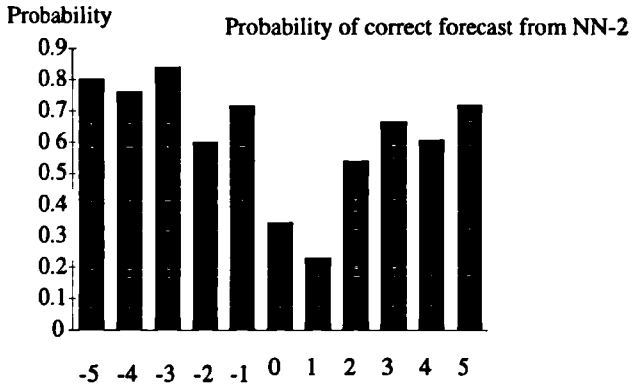


Figure 8.3.4: Histogram of probability of correct LGFC price trend forecast, (NN-2 for the period 18/11/82 to 27/8/85).

In Figure 8.3.4, (using £1 price bands) we can see that NN-2 performs badly when the LGFC price rises in the region of 0-1, and 1-2 pounds. For all other categories of price movement (including the corresponding price falls of 1 pound), NN-2 scores above 50% in terms of a forecast. To improve the NN-2 model we wish to generate a new model that improves the NN-2 probability profile over all price range movements in the LGFC. To do this, we first consider what additional data is available for the GA to model. In the training of NN-2 over the period 18/11/82 to 27/8/85 the NN-Validation module within ANTAS stores a range of analysis data sets. These are depicted in Table 8.3.1.

The entries in Table 8.3.1 are as follows: **Key** is the number of the experiment (500 in total). **F-mov** is the forecasted price movement of the series. **T-Mov** is the target (moving average) movement of the series over the forecast horizon. **R-Mov** is the raw price movement over the forecast horizon. **BF-Mov** is the movement in the raw series before the real forecast, as opposed to the moving average forecast [see section 7.4]. **Hit** is a count of whether or not a correct price trend was predicted. On the basis of this we can first compare the mean forecast movement over the 50-day horizon and the mean raw price movement over the same period. The Root Mean

Square Movement (RMS) is given by £2.19, and the RMS-error of the forecast is given by £2.53. This suggests that the model is not reliable in terms of forecasting actual price movement.

Key	F-Mov	T-Mov	R-Mov	BF-Mov	LF-Mov	Hit
20	-0 24381	-0.22962	-0 28125	0.625	-0 47006	1
21	-0 54689	-0 36142	0.1875	0 71875	-0 62403	0
22	-0.42442	-0.4674	-0.125	0.03125	-0.78278	1
23	-0.3411	-0 57608	-1.03125	0.125	-0.9517	1
24	-0.34767	-0.65353	-2.0625	-0.84375	-0 95453	1
25	-0.80909	-0.67799	-2.25	-1.53125	-1.03198	1

Table 8.3.1: Data produced via Neural Network Validation Module.

Beyond calculating the exact forecast error for NN-2, Table 8.3.1 provides new information which can be used in the manner described in 6.3.2. That is to say we can use the GA to formulate a model based on Table 8.3.1 in an attempt to correct the bias in NN-2 given in Figure 8.3.4. The GA's first task is to formulate rules of the following kind;

GA-Rule 1:

```
if{
        op(BF_Mov ,t1) then s = s + w1;
        op(F_Mov, t2)  then s = s + w2;
}

if{
        op(s,t3) then price forecast = rise;
}

else price forecast = fall;
```

where, op() is one of $\{<,>,\leq,\geq\}$, t_1, t_2, t_3, w_1, w_2 are simple thresholds and weights, and s is a score for the rule. For the first model, only the BF_Mov and F_Mov data was considered. The GA's task was to find values for the thresholds, and weights and operators for the rule. Using this method the GA was trained on 300 data points and tested out-of-sample on 200 data points. The GA simply optimised the number of times the forecast produced by the rule was correct. Using this method it was easy to find rules that scored in excess of 80% in sample, but which performed very badly out-of-sample. The best score achieved for both in-sample and out-of-sample was 70.6% and 67.5% respectively. The effect of the rule on NN-2's price rise probability histogram is given in Figure 8.3.5. As can be seen the GA rule (GA-R1) considerably improves the network's response to small price rises in the LGFC, however, the performance of the GA-NN rule is considerably worse in the region of large price falls (-£4, -£3, and -£2).

It should be pointed out that Figure 8.3.5 combines both the in-sample and out-of-sample results for the performance of the GA-R1. On the basis of out-of-sample performance, the GA-R1 is rejected as being an improvement over NN-2. The reason for this is the fact that GA-R1 fails to out-perform NN-2 over a 200 day out-of-sample (from their respective training sets) tests. On this basis a hard limit is set by ANTAS in terms of which model should be favoured. Recall that NN-2 scored 73.5% (see Table 8.2.4) on a 200 day out-of-sample test.

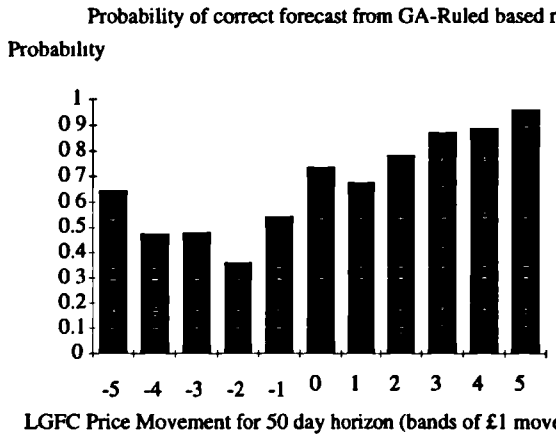


Figure 8.3.5: Histogram of probability of correct LGFC price trend forecast,

(GA-R1 rule for the period 18/11/82 to 27/8/85).

As a comment on this stage of model development it is worth mentioning the overall behaviour of the first set of GA rules that were generated. In most cases very brittle results were produced (for the rules that scored in excess of 80% in-sample, they tended to score below 50% out-of-sample). This may suggest that a more robust means for scoring each of the rules may be beneficial. Rather than attempting to adjust the format of the rules and the fitness scoring for the GA on the reduced data presented above, a more comprehensive approach is to allow the GA to have access to more data, particularly in the form of secondary data series. This will allow the GA to formulate the best model on the basis of all information.

In the next section we start Phase II of the modelling process where we extend the data available for modelling to include secondary data sets. Phase I of the ANTAS modelling has therefore produced NN-2 as the candidate model for forecasting price trends in the LGFC. It is this model that the secondary modelling will attempt to beat.

8.4 Phase II - Secondary GA-RB Models

In this phase of model construction we turn our attention to the secondary data sets that may be influential on the LGFC price series. The data series in question were first introduced in section 7.3 and consists of the LGFC traded volumes, the Over 15 Year Stock Index, All Stock Index, and the 5 to 15 Year Stock Index.

One method that might be considered is to introduce a GA rule model that specifically targets the areas in which NN-2 performs badly, that is to say, a rule that adjusts the NN-2 model for LGFC price movements of 0-1 and 1-2 pound rises (i.e., a model that targets the areas in which NN-2 performs badly). However, whilst this method may have a direct appeal, there is a fundamental problem, namely: if we exclude the poorest region of NN-2 performance from the model (price rises of 0-1) the model achieves a score of 71.23% as compared to 60.4% for the whole model. Therefore if we generate a rule that can specifically identify regions of £0 to £1 price rises so that we can exclude this region form the NN-2 model, we will require a rule that scores an out-of-sample results of over 84.7% so as to beat the original NN-2 model. That is to say, if the rule is to correctly identify when to over-ride the NN-2 forecast it must be able to

identify price rises of between 0-1 with a probability of greater than 0.847. Only in this way will the combined model (with probability 0.846x0.7123 of being correct) be greater than the original NN-2 model (which has probability of 0.604 of being correct).

A score of 84.7% correct seems too high to achieve in practice. Indeed, experiments confirmed this. It was possible to generate rules that could identify a £0-£1.5 rise in the LGFC with 86% in-sample, but which could only score 70% out-of-sample. On this basis, the new model would perform at an expected success rate of 49% (i.e., multiplying both success rates together), which is considerably worse than the existing NN-2 model. This point serves to highlight the general difficulty in generating a GA model that "corrects" the NN-2 model. In all cases, the new rule will have to perform at an extremely high success rate so that a switching process between the NN-2 output and the new model has a better overall chance at predicting the price trend.

An alternative to generating a GA-rule that corrects the NN-2 model, is to use a direct GA approach in order to find a new model that is predictive of the price trend in the LGFC for the forecast horizon. However, this would imply that to combine the new model with NN-2 we would have to introduce some form of voting between the models. If we do this, we will effectively restrict the combined model to making forecasts where the two models agree, which will consequently reduce the overall application of the model. Packard [Pack90] has used this approach and refers to areas of predictability. However, at this stage we have no desire to limit the model's usage, and would prefer to produce a model that always makes a prediction, and that if it is not possible to find a more robust model than NN-2 we shall select this model for the complete out-of-sample tests.

8.4.1 Secondary Model Control Module

ANTAS's secondary modelling avoids some of the difficulties mentioned above by using the GA-Rule Base Module (GA-RB) to probe possible relationships between the target series and new data sets. The objective is to i) find possible data sets that are worth testing via multivariate neural network analysis, and ii) generate new models that are predictive of the LGFC directly (i.e., the GA rules generated).

In section 7.3 we introduced 4 new data sets that may be influential on an LGFC price model. The first stage of secondary modelling within ANTAS is to use the GA-RB to interrogate this data so as to find possible relationships which may be beneficial in terms of forming a predictive rule (as in section 7.5). On the basis of these results a new neural network model may be generated which combines the new data series, provided the model out-performs the existing NN-2 model. In order to test a secondary GA-RB model the rule given below were tested.

Below, op() is one of $\{<,>,\leq,\geq\}$. The simple rule looks to find a trend in the secondary series that in conjunction with a trend in the LGFC is predictive of a price rise or fall over the 50-step forecast horizon. The GA's task is to find values for t_1,t_2,t_3,t_4 and to identify which of the four operators should be used in the final evaluation. The value t_0 refers to the latest time step that the forecast is to be made from. The values of t_1,t_2 range over all possible time steps within the

training data. In all cases the best results were achieved when $t_3 = t_4 = 0$ and op() was >, the final expression in each case to being ($s_1 > 0$ and $s_2 > 0$). A GA used to optimise GA-Rule 2 for all series produced the results shown in Table 8.4.1.

GA-Rule 2:

s_1 = lgfc[t_0)-lgfc(t_1)
s_2 = series[t_0)-series(t_2)

if{
(op(s_1,t_3) and op(s_2,t_4)) then price forecast = rise
}

else price forecast = fall;

GA-Rule 2 for secondary series	
LGFC Traded Volumes Rule formed: t_1=250, t_2 =201. In-sample score (300 data points): 80.3%. Out-of-Sample Score (200 data points): 63% Performance Histogram: see Figure 8.4.1.	Over 15 Year Stock Index Rule formed: t_1=141, t_2 =210. In-sample score (300 data points): 88%. Out-of-Sample Score (200 data points): 60.5% Performance Histogram: see Figure 8.4.1.
All Stock Index Rule formed: t_1=141, t_2 =210. In sample score (300 data points): 88.0%. Out-of-Sample Score (200 data points): 60.5% Performance Histogram: see Figure 8.4.1.	5 to 15 Year Stock Index Rule formed: t_1=142, t_2 =210. In sample score (300 data points): 88.3%. Out-of-Sample Score (200 data points): 60.5% Performance Histogram: see Figure 8.4.1.

Table 8.4.1: GA-Rule 2 scores for secondary series.

As a control GA-Rule 2 was also used for the LGFC. The results for this rule are given below:

LGFC price series.
Rule formed: t_1=142, t_2 =210.
In sample score (300 data points): 87.6%.
Out-of-Sample Score (200 data points): 68.5%
Performance Histogram: see Figure 8.4.2.

There are several interesting facts to note about the above results. Firstly, the same rule in almost all cases has been generated. For all of the series, bar the traded volume, two trends ranging from 141 days and 210 days are used. The 141 figure is almost always applied to the LGFC and all other series, bar the traded volume, which used 210 days preceding the forecast date. All score at virtually the same level (around 88%) with the exception of the traded volume (80.3%). The traded volume is also distinguished by the fact that it does worse than the GA-rule 2 applied to LGFC alone (87.6%). However, the traded volume scores the highest out-of-sample with 63% correct, compared to 60.5% scored by the others. Finally the LGFC control rule scores best out-of-sample, with a score of 68.5%. In fact the LGFC control rule performs almost at exactly the same level as the best other rules in-sample, and performs the best out-of-sample. In terms of ANTAS this terminates the secondary modelling phase. That is to say, on the basis of out-of-sample forecasting the best model is simply a univariate model of the target series, with NN-2 being the candidate model (i.e., its performs best over a 200 out-of-sample test). We shall

analyse the decision process within ANTAS in section 8.6. In particular, we shall run a number of control experiments in order to assess the relative success of the various techniques that have gone into selecting NN-2 as the candidate model for forecasting price trends in the LGFC.

8.5 Phase III - Validation and Simulated Live Trading

The model construction phases within ANTAS have now been completed. What remains is a genuine test of the candidate model under live conditions. NN-2 is the candidate model produced by ANTAS. The system has examined a number of alternative models, but on the basis of in-sample validation results NN-2 has scored the highest hit ratio, with a 73.5% score on a 200-experiment test set and 60.4% on a 500-experiment test set. In the final phase NN-2 is subjected to 1000 out-of-sample experiments. This data corresponds to the period 31/10/86 to 16/10/90 for the LGFC price series. None of this data has been included in any of the earlier experiments, and for the final set of experiments no parameters of the model were adjusted, with all results being achieved in a single run. The ANTAS system model of the LGFC is the following:

ANTAS - LGFC Price Trend Model Configuration - NN-2	
Architecture	15-4-1 Fully Connected
Forecast Horizon	50 Steps
Moving Average	23 Days
Scaling	0.5
Learning Multiplier	1.1
Learning Divider	2.0
Hidden Nodes	Hyperbolic Tangent
Output Nodes	Linear
Learning Style	Batch
Training Data	250 past values of the LGFC

Table 8.5.1: ANTAS candidate Neural Network model for the LGFC.

For the 1000-day test set NN-2 correctly forecasted price trends in the LGFC with an accuracy of 60.6%. The forecast horizon is 50 days, however since NN-2 uses a 23-day centred moving average this represents a genuine forecast of 50-12.5=37.5, or 37 days for the raw price series. This is an extremely good result and indicates that NN-2 has isolated a reasonable level of determinism associated with price trends within the series. For the absolute values of price changes NN-2 does considerably worse with an average error (as measured against the moving average) of -0.56 compared with a mean price movement of 0.002. The full results are summarised in Table 8.5.1.

For Table 8.5.2 recall that Target refers to the 23-day moving average of the LGFC that NN-2 is trained on. As mentioned Table 8.5.2 shows that NN-2 does not perform well in terms of finding an absolute price movement for the target moving average series. However, NN-2 was intended as a means for forecasting price trends in the LGFC (direction of price movement) and 60.6% is an extremely good result compared to random chance. From Table 8.5.2 the probability of a price rise for the raw series is 0.496 which, using the same statistical arguments as in 8.2.3, we should accept as an estimate of the true probability of 0.5. This compares with a probability of 0.606 that NN-2 correctly forecast the LGFC direction. For 1000 experiments this gives a

statistical confidence of over 99.99% that NN-2 did not achieve this result by chance (with a z statistic of over 6.7 - taking $\overline{p} = \overline{q} = 0.5$ and using Equation 8.1).

NN-2 Out-Sample Results 50 day forecast for the LGFC (1000 experiments)	
Mean Target Movement	0.002
Mean Forecast Movement	-0.5575
Mean Error	-0.56019
Mean Raw Movement	0.01
Variance Target Movement	12 0034
Variance Forecast Movement	3.21
Correlation Target and Raw Movement	0.96
Correlation Target and Forecast Movement	0.21
Correlation Forecast and Raw	0.21
% of Price Rises - Target	48.8%
% of Price Rises - Raw	49.6%
Forecast Target Direction	60.4%
Forecast Raw Direction	60.6%

Table 8.5.2: Results for NN-2 on 1000 Out-of-Sample Forecasts.

One apparently surprising statistic to emerge from Table 8.5.2 is the fact that NN-2 appears to forecast Target trends with slightly less accuracy than Raw price trends (60.4% compared with 60.6% respectively). This fact is explained once it is noted that the Target forecast period corresponds to 50-days of the moving average, compared with a forecast period of 37 days for the raw price series (i.e., genuine forecasting of the raw series takes place from the last available price for the LGFC). If this extra period of 13 days is taken into consideration then it is possible to improve the forecast accuracy for the moving average [King94]. However, it should also be noted that NN-2 was chosen on the basis of validation results that related to its ability to forecast raw price trends, as this is our ultimate goal.

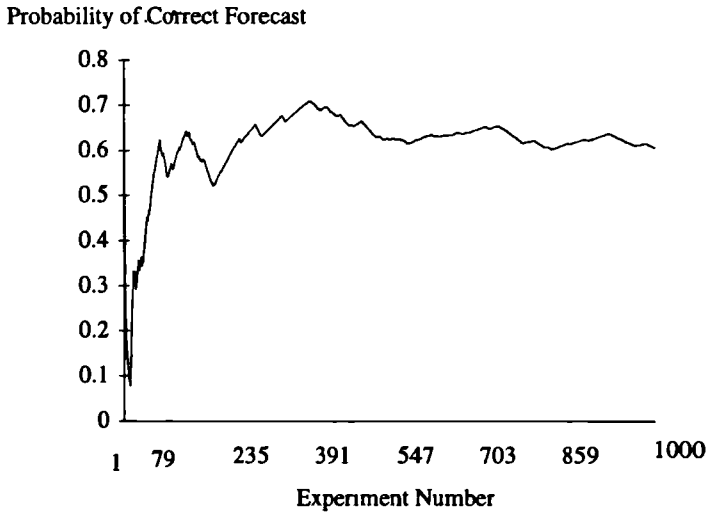


Figure 8.5.1: Cumulative Probability for a NN-2 correct forecast.

To analyse the performance of NN-2 more closely Figure 8.5.1 shows the probability of a correct forecast for NN-2 as the number of experiment increases. As can be seen NN-2 performs

badly for the first 79-85 experiments, after which the probability level starts to stabilise around the 60% level.

To get some idea as to the stability of NN-2's performance in comparison with the in-sample tests, and to see if NN-2 retains aspects of its in-sample performance Figure 8.5.2 provides the probability histogram of correct price forecast as compared to price movement within the raw LGFC price series.

In Figure 8.5.2 we get a similar profile of performance to that given in Figure 8.3.4 for NN-2's in-sample performance over 500 experiments. The main changes occur at the 0-1 price rise where NN-2's out-of-sample performance has considerably improved (to just under 50% correct compared with less than 40% correct for the in-sample results). Where NN-2 appears to have performed badly compared to the in-sample results is the 3-4 price rise band (with a score of below 45% compared to the in-sample 60%+ performance).

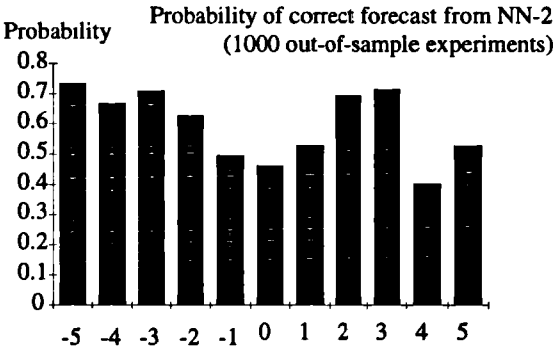


Figure 8.5.2: Histogram of probability of correct LGFC price trend forecast.

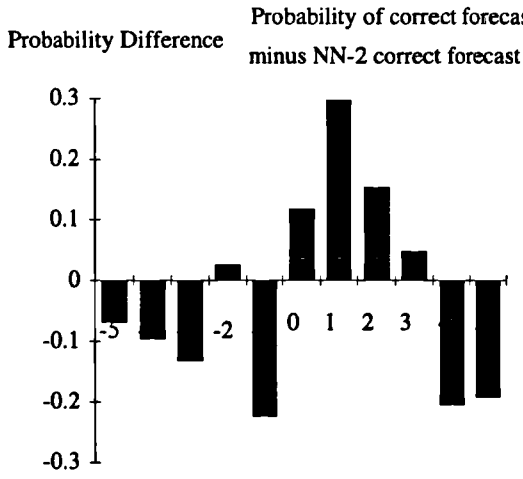


Figure 8.5.3: Probability changes for NN-2 out-of-sample compared to in-sample.

To investigate the stability of NN-2's performance out-of-sample compared to in-sample, Figure 8.5.3 depicts the probability change over each of the price bands given in Figures 8.5.2 and 8.3.4. As can be seen, there are large probability changes in favour of the out-of-sample performance around the 0-3 price rise area, with a corresponding fall in probability in the areas of large price falls (-2 to -5) and large price rises (3-5). The mean difference between the two

performances is -0.024 across all bands, representing a fall in performance out-of-sample compared to the in-sample performance (despite the fact that the overall success for NN-2 is better out-of-sample, with 60.6% compared to 60.4% for the in-sample). This analysis may suggest that despite NN-2's good overall performance its stability may be questionable.

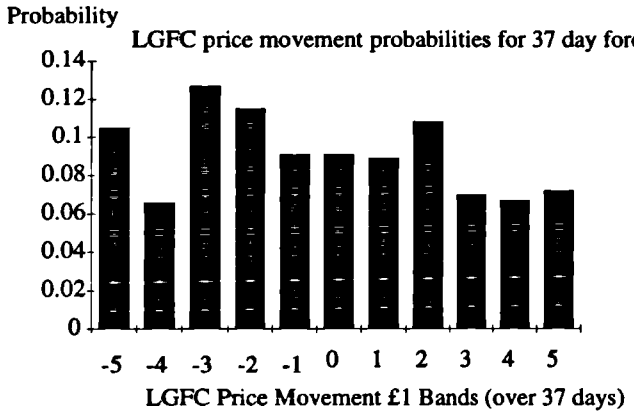


Figure 8.5.4: LGFC raw price movement over the NN-2 forecast horizon (37 days).

Figure 8.5.4 gives the price movement for the LGFC over NN-2's forecast horizon. This probability histogram gives a measure of the price movement frequency for the LGFC. As can be seen price movement within discrete price bands is fairly uniform, suggesting that a forecast of absolute price movement may be very hard to achieve.

NN-2's score of 60.6% contradicts all forms of the Efficient Market Hypothesis (EMH). According to EMH it should not be possible to achieve this level of forecast accuracy, particularly when we considered that NN-2 only takes into account past price movements. Moreover, the statistics for price movement in the LGFC suggest there is no obvious bias (i.e., price trend) that NN-2 could be using as a means for achieving this score. Moreover, Figure 8.5.3 does not suggest that NN-2 has an obvious bias to forecasting price rises as opposed to price falls. All of which gives a credibility to the modelling result. In section 8.6 we shall investigate a number of aspects relating both to ANTAS decision process as well as the possibility of finding possible explanations for the exceptional performance of NN-2.

As a final step in the analysis Figure 8.5.5 provides the Equity Curve for NN-2 traded on the LGFC. The Equity Curve is a standard trading method [Refe95] for analysing the performance of a market model or trading strategy. It shows the cumulative return from trading as dictated by the trading strategy, in this case we buy (go long) if NN-2 forecasts a price rise, and sell (go short) if NN-2 forecasts a price fall.

The significant aspect of Figure 8.5.5 is the consistency and relative smoothness of the profit curve. What traders assess from the Equity Curve is the manner in which profit is made, i.e., an equity curve with many peaks and troughs has many losing trading periods. A smooth Equity Curve is regarded as far more desirable than a high net return (long periods of loss trading usually result in a trader being sacked, even if at some stage in the future their strategy would be vindicated). The largest *drawdown*, for NN-2, where drawdown is the greatest percentage

retracement of the cumulative returns [Refe95], is 11.4% (for the period 26/6/90 - 29/8/91). This figure is acceptable [Refe95] and in conjunction with the overall characteristics of the Equity Curve suggests that NN-2 could survive as a basic trading strategy. However, at this stage it is worth recalling that the LGFC price series is artificial, and that in many instances there is a choice of three contracts which the series may be referring to. This suggests that some additional analysis would be required in order to use the information generated by NN-2 to realise a return.

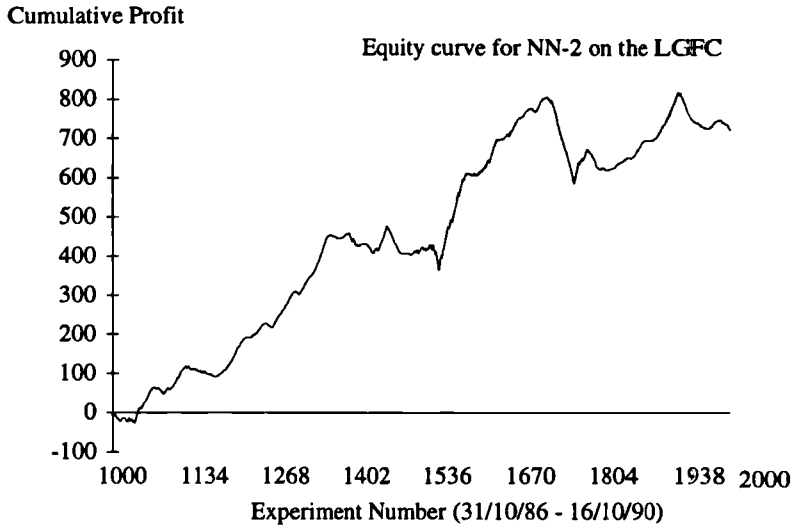


Figure 8.5.5: LGFC equity curve for the 1000 days experiments.

8.6 Controls: Analysis of ANTAS

The results reported in the previous section would appear to justify the ANTAS model design process. In this final section we take a closer look at the design process and run some control experiments in order to validate the model selection used by ANTAS. In particular we are interested in justifying the lengthy steps that were used in designing the ANTAS model. For example, it may be the case that an arbitrary neural network could have achieved the same result as NN-2 in forecasting the LGFC, or that a standard GA could have been used when optimising the rule-based approaches. In short, this section provides some additional experimentation in an attempt to explore the benefits offered by ANTAS.

8.6.1 Choosing a Network Architecture

For ANTAS we developed Network Regression Pruning (NRP) in order to identify a neural network architecture for a given time series modelling problem. As has been remarked on a number of occasions, in principle a large architecture should have access to the same functionality as a small network architecture. In Chapter 5 we presented evidence that for good generalisation a smaller network (preferably Minimally Descriptive) increases the likelihood of good out-of-sample performance. To test this for the LGFC price series the following neural network was trained for 500 contiguous values of the LGFC price series.

NN-Control Test	
Architecture	20-10-1 Fully Connected
Forecast Horizon	50 Steps

Moving Average	23 Days
Scaling	0 5
Learning Multiplier	1.1
Learning Divider	2 0
Hidden Nodes	Hyperbolic Tangent
Output Nodes	Linear
Learning Style	Batch
Training Data	250 past values of the LGFC

Table 8.6.1: A control neural network configuration as a comparison to NN-2

The control neural network has the same parameters as NN-2 bar the network architecture. The results of 500 experiments with this network was a raw price direction prediction accuracy of exactly 50% i.e., the network perfomed no better than random chance for each of the forecasts. This provides very strong evidence in favour of the multiple techniques applied within the thesis in terms of designing NN-2. The fact that a change in the architecture should decrease the performance by 10% once more underlines the importance of finding a good neural net model.

8.6.2 GA Control Tests

One aspect that was raised in Chapter 4 was the use of multiple representations in the GA (all GAs within this have used base 2, 4 and 16 dynamically in the optimisation process). The simplicity of GA-Rule 2 (section 8.4.1) provides a good opportunity to test the GA design used in ANTAS for the LGFC and related series as compared to the standard binary GA. As a test of the multiple alphabet GA a comparison was made between the standard binary GA and the multi-representation GA for GA-Rule 2. Figure 8.6.1 gives the results for finding a predictive rule using the LGFC and the All-Stock Index.

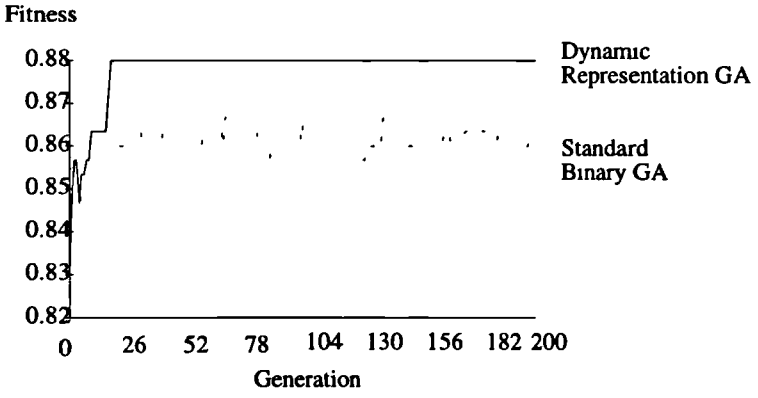


Figure 8.6.1: Dynamic Representation GA and Binary GA for GA-Rule 2 for the All Stock Index.

The result shown in Figure 8.6.1 is typical of the rule generation for all the other secondary data series (LGFC-Traded Volume, 5-15 Year Stocks and Over 15 Year Stocks). In all cases the binary GA failed to find the optimum found by the dynamic representation GA. Figure 8.6.1 shows the convergence profile for the binary GA as compared to the dynamic representation GA for the All Stock Gilt Index. As can be seen the standard binary GA achieved a score of just below 87% compared to the dynamic representation GA which scored 88% (both GAs used the same starting population, and the same mutation, crossover and truncated selection rates). What is interesting about Figure 8.6.1 is the instability of the binary GA. The mutation rate was set as

$1/L$ for L the length of the binary string [Chapter 4]. This result would once more appear to vindicate the GA design used by ANTAS.

8.6.3 Second Order Modelling

One of the key decisions made by ANTAS was the rejection of secondary data series in terms of finding a model for the LGFC. In section 8.4.1 all secondary models (based on LGFC-Traded Volume, All Stock Index, 5-15 Year Index and Over 15 Year Index) were rejected on the ground that none out-performed the LGFC GA-RB model out-of-sample.

In terms of the decision process within ANTAS the suggestion was that a univariate model should be used. This was simplified by the fact that during the Primary Modelling Phase a univariate neural network model for LGFC had been developed (namely NN-2). Moreover, this model had out-performed all of the rules-based models generated by the GA. In this section we would like to test this decision process, in particular we shall use the ANTAS modelling procedure to generate a multivariate neural network model for the LGFC and analyse the level of performance.

On the basis of the results of section 8.4.1 we see that the best secondary model makes use of LGFC traded volume data (this scored 63% out-of-sample compared to 60.5% for all other models - Table 8.4.1). Using NRP to infer an LGFC-Volume neural network architecture resulted in a suggested complexity of 43. From this two test networks that combined the 15-4-1 network for the LGFC were generated. These are depicted below:

Multivariate NN-Control	M-NN-1	M-NN-2
Architecture	30-6-1	25-8-1
Training Data 1	15-LGFC Price Lags	15-LGFC Price Lags
Training Data 2	15-LGFC-Vol Lags	10-LGFC-Vol Lags
Forecast Horizon	50 Steps	50 Steps
Moving Average	23 Days	23 Days
Scaling	0.5	0.5
Learning Multiplier	1.1	1.1
Learning Divider	2.0	2.0
Hidden Nodes	Hyperbolic Tangent	Hyperbolic Tangent
Output Nodes	Linear	Linear
Learning Style	Batch	Batch
Training Data	250 past values of both series	250 past values of both series

Table 8.6.3: Multivariate Neural Networks for LGFC and LGFC Traded Volume.

M-NN-1 and M-NN-2 were tested on 500 experiments corresponding to the 500 tests used for NN-2 for the in-sample data (Phases I and II). For this number of experiments M-NN-1 scored 56.4% and M-NN-2 scored 58.8% (as compared to over 60% for NN-2). Both these results represent a statistically significant fall in performance compared to NN-2 and again suggest that the ANTAS decision process was valid.

8.7 ANTAS: Conclusions

There are a number of aspects worth underlining in terms of the ANTAS design and the results that have been achieved in the context of the some of the earlier work presented within

this thesis. Chapter 3 detailed a number of problems associated with the design of a neural network for a given learning problem. The central difficulty relates to the parameterisation of a network model so as to achieve good generalisation. One factor that appears to have worked well in ANTAS is the ability to identify a good neural network architecture for a given problem. The advantage can be attributed to the technique introduced in Chapter 5, namely Artificial Network Generation (ANG). ANG provided the means by which a controlled series of experiments testing network architectures against generalisation could be conducted. It established a bias towards minimally descriptive network designs. Moreover, ANG provided a means by which an architecture for the network could be inferred when used in conjunction with Network Regression Pruning (NRP). As was shown in 8.6.1 the LGFC does not provide an easy series to model, and a network of higher complexity (than suggested by NRP) failed to find any level of determinism within the LGFC series. Another factor that worked well was the method of model comparisons, in which a price probability profile is used across the spectrum of a model's performance. The technique rejected the use of additional secondary data series, and tests conducted in 8.6.3 would appear to vindicate this decision.

One aspect of ANTAS that has been less successful is the combination of GA and neural network. Despite the fact that the Dynamic Representation GA (DRGA) out-performed the standard Holland GA in all of the tests conduced in both Chapter 4 and this Chapter, the combination of neural network and DRGA failed to isolate a single network design for the LGFC. For the most part this was caused by the computational resource required to train multiple neural networks. As has been shown in this Chapter, if the GA is used to adjust a network's training parameters for a fixed training set then over-training occurs. Only by using randomly selected training sets could we hope to avoid this problem, and this proved too computationally expensive to be carried out in practice. However, the DRGA performed well, in comparison to other Multi-Representation GAs and the standard GA, on all smaller problems that were tested. Moreover, the results of section 8.2.2 suggest that the DRGA worked *too well* in terms of setting the neural network parameters for a single training set., in that it produced an over-trained network tunned to the single data series. NRP was responsible for the network design used by ANTAS, and this may suggest that ANG should be used to test neural network training parameters against generalisation (as opposed to architectures), and that this may be one way of narrowing down further the search for the best network design.

8.8 Summary

This chapter has presented a detailed account of ANTAS applied to the LGFC price series. We have shown how ANTAS set about designing a neural network model for this price series. It has been shown how a univariate model was hypothesised and how secondary data was rejected in forming the ANTAS neural network model for the LGFC. ANTAS eventually produced a neural network capable of predicting price trends in the LGFC with over 60% accuracy. This result was achieved over 1000 out-of-sample experiments. It has been shown that the probability of scoring this result by chance is less than 0.0001. Having presented the results of ANTAS a number of control experiments were used to validate the ANTAS design process. Firstly it was

shown that an arbitrary architecture for the neural network could not model the LGFC. Secondly it was shown that multiple alphabet GAs out-performed the standard binary GA for the rule-based modules. Finally, it was shown how more complex models than those proposed by ANTAS performed significantly worse over the LGFC data series. In total this suggests that isolating any level of determinism within the LGFC is a considerable achievement. Moreover, the fact that ANTAS scored over 60% provides strong evidence in favour of the automated design process offered by ANTAS.

Chapter 9

Summary, Conclusions and Future Work

This final chapter provides a summary and a critical review of the work contained within this thesis. In particular we highlight the research contributions that have been made and discuss aspects that should lead to future work.

9.1 Thesis Motivations

In this thesis we have designed and tested a system for financial time series prediction. The overall objective has been the development of an automated system for time series analysis. Underlying this objective has been the analysis, experimentation and development of methods for both understanding and applying two adaptive computational techniques, namely feed-forward Neural Nets (NNs) and Genetic Algorithms (GAs). In the next few sections we review the progress of this thesis in terms of meeting these objectives and in terms of the conclusions the work offers. We provide a critical analysis of the work presented and discuss aspects that form a basis for future work.

9.2 Objectives: Neural Networks and Learning

Feed-forward Neural Nets (or Multi-Layer Perceptions, MLPs) are classed as a machine learning technique with the ability to learn from examples. In Chapter 3 we examined this class of neural net and showed that in reality feed-forward neural nets more precisely resemble non-linear regression techniques. That is to say, this class of neural net has a strong overlap with traditional statistical modelling methods, and that despite the universality of this modelling paradigm (i.e., the fact that MLPs are universal functional approximators) there are considerable technical problems associated with the parameterisation of the model in order to achieve good results. It was suggested that at present most neural net applications require careful experimentation on behalf of the human designer, and that to develop a successful neural net application requires considerably more effort in terms of design than simply training a network on past data.

In Chapter 3 we investigated the detail of the neural net parameterisation problem describing the effect of various parameter choices on the output of the trained network. We suggested that the ultimate goal of any learning system is generalisation, and that feed-forward neural net training is not directly related to generalisation. In this respect feed-forward neural networks *do not* strictly qualify as learning systems. The fact is that neural net training is only a single step contained within the learning process. Moreover, in traditional earning theory terms the neural net training process is ultimately a process of hypothesis generation in which each trained neural net represents a single hypothesis of the functional relationship contained within the training data. Worse still, is the fact that unless formal restrictions are placed on the type of functional relationship that is being learnt then consistency on historic data is no guarantee of future

success. This formal requirement has been stipulated on a number of occasions within the theoretical learning theory community [Vali84], [Wolp92], [AnBi92], [BEHW89], [AngSm83]. This presents a difficulty for the applied machine learning community, particularly those interested in modelling real-world problems. If we concede that formal restrictions should be placed on the type of learning problem that we attempt to model then this effectively suggests that we should only consider applying machine learning methods to machine learnable problems. However, this tautological luxury is not generally available for real-world problems. In most circumstances we have no guarantee that a real-world process is stable (in terms of a generating process), or that if it has been stable that it will remain stable.

9.3 Thesis Outline and Research Contribution

The dilemma presented in 9.2 represents the current gap between theoretical learning theory and applied machine learning. On the whole few guidelines exist in terms of developing a successful machine learning application, and in practice there is a development cycle in which a neural net, say, is used by a designer to hypothesise a relationship between past examples of a given process. There then follows extensive validation in which the human designer adjusts parameters, introduces new data sets and adjusts controls until either they conclude that the process can not be modelled, or they become satisfied that a genuine model has been found. What has been attempted in this thesis is the automation of this process for feed-forward neural nets, and in terms of what has been discussed above, the goal has been to develop a *learning system*, that hypothesises and validates and eventually forms its own model of a target process.

To achieve this objective we have taken a very pragmatic approach. In Chapter 3 we started with some general analysis of the feed-forward neural net model. It was shown in a very direct way how a feed-forward neural net can be used as a universal functional approximator, and in contrast to the many existence proofs for this property [Cybe89], [Funa89], [Horn91], [HoSW89] we provided an explicit formula for setting the weights and architecture for any given finite data set. This formula provided an explicit demonstration of a how a neural net can be used as a trivial look-up table for a given functional relationship. Such a network would be classed as over-trained, and in no sense would it be likely to generalise out-of-sample. Our next step was to look at ways in which to approach the neural net parameterisation problem so as to avoid over-training and to attempt to improve our chances of achieving generalisation.

It was pointed out that the most obvious way in which to produce a neural net learning system was to automate the validation process, that is to say, to automate out-of-sample testing and formalise the out-of-sample statistical analysis in order to select a single model from a set of hypothesised models. However, it was also pointed out that this approach was unlikely to succeed. The reasons for this was the sensitivity of neural net training to parameter choices. An example was given where a neural net was trained on random data (lottery numbers) and on the basis of out-of-sample performance a network was selected after adjustment of training parameters. It was then shown that such a net appeared to accurately forecast the random series out-of-sample. The purpose of this demonstration was to show how easy it is for out-of-sample

data to *move* in-sample once multiple testing is used. In effect we suggested that the common method of training a network until the out-of-sample validation error rises is flawed and will produce misleading results. Having made this observation it was then suggested that a more subtle use of in-sample data in terms of hypothesising a model was required if over-training was to be tackled at source. The manner in which this was ultimately achieved was presented in Chapter 5. In Chapter 3 it was also suggested that in terms of probing the space of possible neural net hypotheses one method that has been employed by other researchers is the use of Genetic Algorithms (GAs). To explore this possibility Chapter 4 provided a general analysis and examination of this technique so as to formulate a strategy in terms of using a GA to aid the neural net modelling in at least two respects: firstly as a means for selecting data, and secondly as a means for parameterisation. More generally, the purpose of Chapter 4 was to develop an understanding of GA techniques with a view to using GAs as a means for automating the neural net design process.

9.3.1 Multiple Representation Genetic Algorithms using Base Changes

Chapter 4 provided a detailed analysis of the GA search process and introduced a new class of GAs that made use of *Transmutation* and *Transmigration* operators. Both operators relate to randomly adjusting the representation used by a GA during the course of a run and were introduced as a means for avoiding some of the representational bias associated with the static binary GA. In contrast to a number of existing GAs that make use of dynamically changing representations two variations were introduced in Chapter 4. Firstly we employed random changes in the algorithm's representation, and secondly we introduced base changes to remap the space. Random changes were advocated on the grounds that once a search heuristic has halted there is no basis by which to infer what the new representation should be (i.e., the algorithm may have found the global optimum). The use of base changes is also new. Traditional GA theory has held that binary representations are optimal [Gold89], [Holl75], the justification being that binary coding maximises the string length and therefore the number of schemata sampled. In Chapter 4 we showed that this was not the case if all similarities (all hyperplanes as given by new * variables) are taken into consideration (a similar argument is given in [Anto89]). Base changes therefore provide a simple and effective means for remapping the search space, both in terms of adjusting the hyperplane sampling characteristics, and in terms of the actual shape of the space. That is to say, a change in base opens up new connectivities for mutation, and provides new stabilities under crossover. Six different styles of Multi-Representation GAs (MRGAs) were compared with a binary GA and random sampling on a range of standard test functions. The MRGAs out-performed the standard GA on all problems, with the best GA using a single pool with a change in representation (base) across the pool at each generation. This GA was used in all subsequent work within this thesis.

Several general aspects regarding the work in Chapter 4 are worth stating, both in terms of the way in which the strategy for designing ANTAS emerged, as well as for future work in generalised optimisation techniques. The most important aspect of recent research into generalised search algorithms is the fact that all search algorithms are equivalent [RadSu95],

[WolMc95]. In terms of finding a global optimum any two search algorithms can be shown to be equivalent if arbitrary representations of the search space are allowed. For example, it is always possible to permute the encoding of the search space retrospectively so that a given search trajectory in encoding space is forced to lead to the global optimum (i.e., create an encoding defined by permuting the algorithm's original solution point with the global optimum). This has far reaching implications—it says that for a given cost function a search heuristic can always be vindicated by the *correct* choice of encoding. Moreover, it says that an algorithm can never know when it is making a *correct* move, in that no search trajectory can be said to be better than another, since for any search trajectory there exists an encoding that leads it to the global optimum. All search algorithms have been formally shown to be equivalent over all possible search problems [WolMc95], [RadSu95].

What was illustrated in Chapter 4 was the practical manifestations of search equivalence. In essence this gives rise to the fact that the relative difficulty of a fitness landscape is algorithm dependent. It was shown that rather than a complex remapping of space, a simple base change can be sufficient to change the comparative difficulty of a search problem. Two examples were given in which a deceptive problem for a binary GA were made *easy* by a change of base. This underlines the fact that in the absence of *a priori* knowledge about the function landscape (as defined by any algorithm), the choice of operator or representation (real valued, binary, Gray-binary, non-standard, base *A*) is completely open, with no formal means for determining the correct choice. Given this situation a reasonable query might be, if all forms of search algorithm are equivalent why use one at all? In other words use random sampling. There are two immediate reasons are, firstly, little is known about the actual complexity of most real-world search problems, certainly not enough to conclude that in the absence of good partial knowledge of a problem type that search heuristics should be abandoned. Secondly, many researchers have experience of tackling real-world problems with existing search methods and have achieved good results (certainly compared to random sampling). An example of this was given in Chapter 4. The results given in Table 4.4.1 show that on average random sampling had the worst performance of all the search methods. On this basis, techniques that enhance existing search methodology should be encouraged. In this sense *multiple* search heuristics have a pragmatic quality in that they provide a means for employing multiple search strategies to a given problem by dynamically adjusting the algorithm's view of the function landscape. Moreover, in light of the limitations on all forms of search, a *random* sampling of search heuristics applied to a given problem provides an extremely pragmatic means for probing the space of possible search strategies that may unlock a given search problem.

Despite the introduction of multiple representation GAs, Chapter 4 also suggested that the direct use of GAs for neural net parameterisation may be problematic. The reason for this was the sensitivity of neural nets to parameter choices and the sensitivity of GAs to the choice of encodings. The hypothesis was that generalisation at present is poorly formulated in terms of an optimisation process, and that unless careful safeguards are taken then over-training would seem inevitable. This suggestion was vindicated in Chapter 8 when a GA and neural net were

combined in order to design a neural net for the Long Gilt Futures Contract. It was shown that despite the fact that a vastly reduced set of search alternatives were made available to the GA (i.e., the network's architecture was removed from the search process) the network over-trained when left to a single training set. In keeping with the general layout of this summary we shall come back to this issue after we have discussed the work contained in Chapters 5, 6 and 7.

9.3.2 Artificial Network Generation

Once the decision had been made to try to limit the use of a GA in order to parameters the neural network, then some other method for deciding network parameters had to be found. One of the main problems in developing a strategy for neural net parameterisation (particularly an automated strategy) is the lack of formal guidelines within which to work. This problem largely stems from the fact that feed-forward neural nets are universal functional approximators and therefore it is hard to find formally binding relationships between the complexity of a neural net and the likelihood of good generalisation without taking into account i) the problem the network is attempting to map, ii) the probability distribution of the weights used prior to training, iii) the probability distribution of the errors, iv) the method of network training or weight space traversal, and v) the method of network validation. To short-cut all of these considerations this thesis took a very direct approach. One of the prime parameters in designing a neural net application is the architecture of the neural net to be trained. At present there are few guidelines which exist that can be used in order to design a network architecture for a given problem, and to the author's knowledge there has been no systematic analysis of the affect of different architecture designs on a network's ability to generalise. To address this issue and to investigate the effect of a network's architecture on the likelihood of good generalisation this thesis introduced a method for testing a network's architecture against generalisation. The techniques developed was Artificial Network Generation (ANG).

ANG uses a GA in conjunction with a network architecture in order to generate a learning problem, or in this case a time series, with a known approximately minimally descriptive neural net generating process. What ANG provides is a means for directly testing i) a neural net architecture, ii) a neural net training process, and iii) a neural net validation procedure, against generalisation. The reason for this is that all three factors can be adjusted and the results can be compared with the known solution (i.e., the generating neural net architecture). To use ANG as part of a systematic analysis of a neural network's architecture it was important to establish that the technique gave rise to a minimally descriptive generating process. This was tested by using networks of smaller complexity than the generating network and analysing the Mean Squared in-sample Error. This was tested for one of the artificially generated series and the results were presented in 5.3.3. This experiment confirmed that for the 12-7-1 generating network all smaller networks failed to match the MSE on the training set compared to 12-7-1 network. Ideally this experiment should have been conducted for all the ANG series. However, 5.3.3 did suggest that a GA could be used to approximate a Minimally Descriptive Network for the generated series. Having established the technique as sound, the next phase was to use ANG as a means for probing the relationship between architectures and generalisation.

The experiments of 5.3.4 used ANG to test the hypothesis that a network of the *correct* topology (i.e., the one used to generate the time series) generalises better than a larger network (i.e., one that has complexity much larger than the generating process). Using five generated series for five different network topologies we tested this hypothesis. In each case we trained two networks (one large architecture, one correct architecture) 30 times for each series, with each starting from different initial weights on the five ANG time series. The results of this experiment provided statistical evidence to support the fact that a neural net architecture of approximately the same complexity as the generating process generalises better than a larger network. For a one-tail test measuring the difference between the Mean MSE of the out-of-sample forecast for both networks over the five series gave a statistical significance at the 70% level that the smaller (or correct size) complexity generalised better than the larger network. These results therefore suggested that Occam's Razor should be used as a means for improving the likelihood of good generalisation.

9.3.3 Network Regression Pruning

Having presented evidence in favour of Minimally Descriptive Networks (MDNs) in terms of generalisation the next phase was to find a method for delivering MDNs in an automated manner for a given application. One way in which to approach this problem was to use a pruning method. Currently there are a number of different styles of pruning techniques, however this thesis introduced a new method, and one that differs considerably from existing approaches. Network Regression Pruning (NRP) was introduced and favoured over existing methods for a number of reasons. Firstly, NRP differs from existing techniques in that it attempts to hold a network's mapping fixed as pruning is carried out. This was considered an advantage in that it provides a way of exploring the space of possible architectures independently from the mapping that a network has found. In this sense redundancy in the network's architecture can be found, in that NRP explores the space of possible architectures that can approximate a given mapping. The second reason that NRP was considered an advantage is that it does not introduce a validation set, or subjective analysis, in order to isolate the candidate network architecture. NRP is defined simply in terms of the network's performance on the training set. The final advantage offered by NRP is that it provides a means for exhaustively pruning a given network. Ideally we would like to search the space of all possible network architectures and weights in order to select a candidate mapping. However, a complete search is computationally intractable. What NRP provides is a means for exploring the space of architectures via shallowest ascent through weight space based on the Mean Squared in-sample Error. This is made possible by the fact that NRP attempts to hold a given network's mapping fixed.

The output from NRP is a pruning error profile. The pruning error profile is a history of the MSE for the in-sample performance of each of the networks produced via pruning. The hypothesis behind NRP is that as pruning proceeds there should be some point (in complexity space) at which a small network can no longer approximate the given target mapping and that the MSE should significantly rise from that point onwards. For all experiments conducted the pruning error profile conformed to this shape. In order to use this information in an automated

neural network architecture design strategy, some form of trigger to isolate the candidate network was also required. To do this ANG was used to retrospectively fit an appropriate complexity trigger. To lessen the chance of over-fitting the small number of NRP experiments, only simple triggering methods were considered. The simplest method is to look for statistically significant rises in the MSE in the pruning profile. The idea being that a threshold of significance should exist which marks the point at which a small network can no longer approximate the target mapping. Using the variance of the in-sample Mean Squared Difference Error (MDSE) as the threshold of significance produced a trigger that correctly identified the complexity of all ANG time series to within 2 hidden nodes (bar experiment 4). Experiment 4 produced an error of between 3 and 4 hidden nodes. To improve these result a second trigger was introduced that used a dual threshold on the MDSE, that was calculated on the basis of rises in the MSE up to the weight that was being removed (i.e., as opposed to a mean and variance calculated on the full pruning error profile). A dual threshold was therefore introduced so that the lower threshold would mark the boundary of statistically significant rises in the error profile, and an upper threshold would exclude the increases in error caused by an already catastrophically failing network (i.e., the trigger should isolate the point at which the network fails rather than all the networks that fail). Taking this approach it was possible to isolate network complexities to an error that was within 1 hidden node for all ANG series bar Experiment 4, which was within 2 hidden nodes.

The simplicity of the triggering method and the accuracy of the results it produced provided the means for an automated network architecture design strategy and was the method used within ANTAS. There are a number of open issues surrounding the use of both ANG and NRP that are worth exploring. We shall discuss some of these lines of research that both techniques provide in a later section.

9.3.4 ANTAS and the Long Gilt Futures Contract

Chapter 6 presented the full specification for ANTAS. ANTAS combined the analysis and results of Chapters 3, 4, and 5 in order to define an automated design protocol for a neural net time series analysis application. The task of finding a good neural net model was modularised (in terms of the system design) and phased (in terms of the model construction process). We introduced two levels of time series analysis, Primary Modelling and Secondary Modelling. Primary Modelling was restricted to univariate time series analysis, and Secondary Modelling expanded a primary model to include multivariate analysis. In order to infer secondary models Chapter 6 also introduced the idea of using GA-Rule Based (GA-RB) models as a means for hypothesising the likely effect of a secondary series on the target series. The idea was to find a rapid means for hypothesising multivariate relationships before the lengthy process of neural net design was conducted. A recurring problem in the use of neural nets is the time required to design, parameters and validate a model. What the GA-RB modules provide is a fast method for testing non-linear relationships between determinants in order to limit the number of neural net models that are inferred. Moreover, the purpose of the primary model for the target series is to act as a benchmark by which all secondary models are judged. It is worth emphasising at this point

that the ultimate goal of ANTAS is generalisation, and that two levels of validation within ANTAS act to compare models. Firstly, validation is used to choose between secondary models, and secondly validation is used to compare the candidate secondary model against the primary target model.

In Chapter 7 the specific financial series used to test ANTAS was introduced. It was shown how the Long Gilt Futures Contract (LGFC) could be reformulated from a contract-based price series into a contiguous price series suitable for time series modelling. Specific data manipulation modules were also introduced to handle moving averages and to infer forecast horizons associated with the use of moving averages of the target series. Chapter 7 also provided some analyses of the target raw price series. It was shown that the probability of LGFC price rise (or fall) was 0.5 which is consistent with the Efficient Market Hypothesis that the series is random.

9.3.5 Results

Chapter 8 presented a step-by-step analysis of the ANTAS modelling process as applied to the LGFC. Accordingly this chapter consisted of three main phases of model development. Firstly Primary Modelling in which a univariate model of the LGFC was constructed. Secondly the Secondary Modelling phase in which new data was introduced in an attempt to improve the performance of the primary model, and lastly the third phase in which a candidate model for the LGFC was tested extensively out-of-sample. In the final section of Chapter 8 some control experiments were conducted so as to test the design and decision process within ANTAS.

The first phase of model construction within ANTAS produced the first problem within the ANTAS design. NRP had provided a way of infer a small set of candidate network architectures for the LGFC. ANTAS then applied a GA to parameters these models over a number of data sets in order to select the best model. However, due to the prohibitive training times of the neural networks, the method used by ANTAS was to randomly partition the data into training and validation sets. Each network was to score a fitness based on a combination of in-sample training error and out-of-sample validation error over the randomly selected sets. However, the small number of sets chosen failed to provide good convergence profiles for the GAs. In essence the small number of random samples caused the fitness values to oscillate, and did not provide a clear signal as to the best network. When the random sample was changed to a fixed sample a good convergence profile was possible, but in this case the network over-trained on the fixed data sample. This was shown by a network trained out-of-sample from the training data which produced a square wave approximation to this training set. The reason for this was the large values that could be used by the GA for the learning rate multiplier and divisor.

As a consequence of the above problems, the learning rate multiplier and divisor were restricted to much tighter controls. The GA was then used to produce a small number of networks (three) from three separate runs. These networks were then subjected to a large out-of-sample validation trial (200 experiments) in order to select a candidate primary model. In retrospect a better design might have been to fix training parameters and test each of the neural nets produced via NRP directly. Recall, NRP produced 15 candidate networks, which is manageable in terms of

testing all in a distributed manner (i.e., ANTAS provides the scope of running separate NN-slaves on given data sets). A large number of tests could have been conducted in this way, and each network's performance could have been then analysed in terms of its performance. However, despite this set-back ANTAS produced three good neural net models of the LGFC in which the scores on the 200 data set ranged from 68.5% to 73.5% correct. These are extremely good results for a neural net being trained on data that is very close to being out-of-sample (only the forecast horizon has been calculated on the basis of this data). It should also be pointed out that these results were produced on a single run through the data, i.e., the 200 experiments were conducted once without recourse to adjusting any of the network parameters. A score of 73.5% is extremely encouraging considering that this series is judged random on the basis of the statistical analysis of the price movement, and in terms of the Efficient Market Hypothesis. The first phase of primary modelling was therefore complete, with the candidate network consisting of a 15-4-1 fully connected feed-forward network.

In order to complete the primary modelling phase two further levels of analysis were conducted. Firstly the candidate network was tested on a further 500 experiments so a comprehensive analysis of the models performance could be carried out. This analysis provided the basis for further comparisons with the secondary modelling phase within ANTAS. It also provided the basis on which to base a GA rule-based model. ANTAS induced a rule for improving the primary model using moving average price movement (before a forecast) and price movement as predicted by the primary model. The rule failed to beat the primary model in terms of out-of-sample performance (taken over a 200-experiment sample).

Having completed Phase I, the Secondary Modelling phase within ANTAS was analysed. Here four additional data series were considered: the LGFC contract volumes, the Over 15 Year Stock Index, All Stock Index, and the 5 to 15 Year Stock Index. In all cases ANTAS failed to find a model that could beat the primary model. An interesting aspect that did arise from this modelling phase was the fact that the secondary data series that scored the lowest in terms of its correlation with the LGFC scored the best secondary GA-RB model. The LGFC traded volume was the candidate secondary data series in question. In terms of ANTAS this closed the modelling phases and all that remained was the final full out-of-sample testing (1000 experiments), the results of which are discussed below.

9.4 Conclusions

There are a number of conclusions associated with the work contained within this thesis, many of which can be made in terms of the future work that they give rise to. In terms of a final conclusion there are several aspects relating to the control experiments carried out in Chapter 8 which are worth emphasising.

Firstly, the goal was to develop an automated system for applying neural nets to a given time series analysis problem, and to test the system on a financial time series. The results given in Chapter 8 for the ANTAS performance were extremely good. The fact that ANTAS, in a single run, achieved a score of over 60% accuracy on 1000 out-of-sample experiments for a financial

series provides strong empirical vindication for the design. Moreover, all of the statistical analysis of this time series suggested that price trends were random with a probability of 0.5 of a price rise or fall.

The control experiments used to test the ANTAS decision process are also worth emphasising. In Chapter 8 it was shown how both the Multi-Representation GA produced better results than the standard GA for GA-RB modules, and that the system's decision to use the Primary Model held up against full tests on two secondary models. Moreover, it was shown that the design process behind the primary model was also instrumental in achieving the score of over 60%, all of which suggests that ANTAS has been successful in terms of the goals set out at the beginning of this thesis. A final aspect regarding the ANTAS results is that they add to the growing evidence against all forms of the Efficient Market Hypothesis.

There were aspects of ANTAS that clearly failed. The most prominent was the failure to combine neural nets and GAs in order to parameters the neural net. Despite the use of NRP in order to restrict the number of candidate networks, and despite the use of multiple representations within the GA, it was not possible to use the GA to parameters the neural net model directly. The main problem in terms of combining these techniques was the computational resource required to train multiple neural nets. The results given in Chapter 8 also showed the relative ease of over-training once an automated control process is attached to a validation procedure.

In terms of the technical achievements three aspects of the work contained within this thesis warrant special emphasis. Firstly is the ANG technique developed to test neural nets. ANG was instrumental in the design of ANTAS and opens a number of new lines of investigation in terms of neural net understanding (see below). ANG made possible a controlled series of experiments not just in terms of testing neural net design strategies but also in terms of establishing design objectives. For example, the experiments conducted in Chapter 5 provided empirical evidence that Minimally Descriptive Networks should be favoured to larger network complexities regardless of the technique used to find them. Secondly, the pruning method, NRP, developed within this thesis provides a practical method for exhaustively pruning a given network, and bounding a network's complexity for a given learning problem. NRP introduced the idea of fixing a network's mapping as pruning is conducted. This step meant that more information could be extracted from the in-sample data in terms of designing a network, and provides a sensible bound on the complexity of a network that should be used for a given data set. Finally, the third aspect contained within this thesis relates to the multi-representation GAs that were introduced. The use of this form of GA did provide better results in terms of the GA-RB models constructed as compared to the standard GA. Moreover, the GA-RB modules provided an extremely good indicator for the likely success of a non-linear multivariate analysis model. The results of the GA-RBs suggested that ANTAS should keep the primary model and not waste time constructing a secondary model based on any of the four related time series. This decision was again vindicated by a number of control experiments that used a neural net on the LGFC and the LGFC-traded Volume series.

What ANTAS, and the work within this thesis has gone some way in providing is a generalised design protocol for neural network modelling. All of the techniques that have been developed are not restricted to time series analysis and could be used for general pattern recognition problems. There is a general problem with all forms of model building in terms of the design of the model. What has been provided within this thesis is some way of approaching this problem.

9.5 Future Work

There are a number of areas of research raised from the work presented within this thesis, both in terms of clarifying issues with respect the work presented, and in terms of new lines of investigation.

One of the possibilities that has been raised by the introduction of ANG is a much more detailed analysis of the neural network modelling process. For example, a far more detailed analysis of the relationship between a network's architecture and the modelling process would be extremely beneficial to the applied neural net community. ANG raises the possibility of answering questions such as the sensitivity of the correct neural net topology for a given learning problem. Specifically, what is the tolerance of a given neural network's architecture? In this thesis we established a bias towards better generalisation for Minimally Descriptive Networks, however a more comprehensive set of experiments could treat a far wider range of network topologies other than the fully connected networks used in this thesis. It would be of considerable benefit to understand what bias a different topology introduces within neural net training, and the level (with respect to given data set) of error, or probability of error, introduced by a large network complexity, or an incorrect topology. By using ANG it is possible to test much more extensively the relationship between topology, complexity and generalisation. Moreover, ANG could be used to empirically validate a theoretical model of neural net training and generalisation. For example, an extensive set of experiments could be used to find the probability distribution of generalisation error against the complexity of a network.

Related to this is the need for a much better understanding of the pruning process. We introduced NRP in this thesis and made a case for preferring this method over existing pruning techniques. It seems possible that a more analytic approach may be available to understanding both NRP and other pruning methods. For example, it was hypothesised within this thesis that pruning the network and retaining an approximation to a given mapping should bound the complexity of the MDN for a given problem. Using the definition of MDN in Chapter 5 this is certainly true. However, a more analytic investigation could be used to find a more formal relationship between shallowest error ascent and the MDN for a given learning problem. For example, is it possible that NRP could lead to large errors in terms of setting a complexity bound? Or, what tolerance is available for a given learning problem by using NRP? Ultimately, what we would like to know is what room for error is available within the network design? Are there concrete examples where a fully connected network under-performs the correct topology network (i.e., two networks of the same number of nodes but different number of weights)? In

this thesis it was shown that a relatively large difference in network complexities produced a bias towards better generalisation within the smaller network. It may be the case that there is substantial tolerance to errors in the network architecture below a certain limit.

A lot of the above issues relate to the capacity of a neural net [Chapter 3]. At present the VC-Dimension of a neural net is unknown. It is known to be finite and an upper bound has been derived [Haus92]. NRP ultimately attempts to identify the neural net architecture required in order to provide the correct capacity to model a given data series. Clearly, once we focus our attention on a network's ability to generalise, then a much deeper understanding of the complexity (however defined) of a target mapping must be taken into account when fixing a network's size for a given problem. Currently formal bounds on a network's capacity are strictly related to the sample size used to train the network. Using ANG and random samples it would be possible to empirically investigate the relationship between the capacity of neural net architecture, the complexity of the learning problem and the size of the training set. For example, what is the relationship between the complexity (number of weights) of a network and the in-sample MSE of a network trained on a random series? Moreover, NRP could be used to probe this relationship, and provide an empirical bound of the capacity of a network for a given data set size (i.e., the size of network required to map a random series to a given in-sample MSE). Having done this it would then be possible using ANG to empirically test the relationship between the complexity of a given learning problem and the size of the training set. For example, ANG could be used to generate a series with a known neural net solution. We could then systematically vary the size of the training set and complexity of the network (bounded by the network complexity given by the random series). Since we have a known neural net solution (provided by ANG) we can see if there are characteristics of an "under-capacity" neural net that are consistent with a network trained on a random series.

Another issue relating to NRP that was not tested was the order in which weights are removed. This may be a loss of information. For example, it may be the case that certain input nodes are removed early in the pruning process, as opposed to weights being removed uniformly from the complete network. The weight removal sequence might offer clues as to the complexity of the process that is being modelled, and might help in the network architecture design phase. By examining the probability distribution of weights removed it may be possible to derive a better bound on the size and topology of a network for a given problem. Moreover, comparisons between different neural nets (of the same complexity but trained using different initial weights) could be pruned according to NRP, and the order in which weights were removed could be compared. It is possible to conjecture that a Minimally Descriptive Network is far more stable in terms of a pruning profile than larger networks under different initial weights. However, it should be stressed that the ultimate worth of such an investigation would be dependent on some of the issues discussed above. For example, it may turn out that there is reasonable tolerance within a neural net size. In this case spending effort to find the exact topology of the generating network may be a waste of resource.

An issue directly related to NRP is the complexity trigger for bounding the size of the target neural net. In this thesis we argued that we wished to find the smallest network capable of approximating the mapping of a larger network, the idea being that if the larger network over-trained, then the smaller network was open to the same over-training if it could approximate the original mapping. The smaller network therefore offered a bound on the size of network that is needed to map a given data set. In this thesis we used a simple statistical approach to bounding the network complexity. It seems reasonable that there may be a better analytic model of the pruning process (under NRP) that uses the pruning error profiles produced by NRP. If it were possible to characterise the precise equation for this curve it would then seem possible to derive a far more accurate complexity bound. Even using the complexity trigger that was introduced in this thesis, another line of analysis would be to use the network architecture that corresponded to this complexity level during pruning. That is to say, rather than bound the complexity of the network for a given problem, analyse the relationship between the network architecture (as given during pruning) and the given learning problem.

One factor that has been highlighted by the work within this thesis is the similarity between optimisation (or search problems) and learning. In both cases we have a hypothesis space/search space and we must find ways of searching the space of possible solutions in an attempt to find an optimal solution. For search problems it has been shown how deceptive problems will always exist and that one strategy is to use multiple search heuristics in an attempt to unlock the search problem. An analogous approach to a learning problem might be to use multiple-models of the target process. This is a very open area as to how best to combine estimators, and again ANG might prove useful in validating different approaches. For search problems in general there are a number of open questions. For example, we have used multiple search heuristics to tackle a given search problem. How valid is this approach? In a formal sense taking all possible search problems into account it has been shown that this algorithm will not perform better (in terms of finding a global optimum) than any other search algorithm once the space of all search problems is considered. However, is there a bound on the size, or complexity, of search problems that are encountered within the real-world? Or is it possible to derive classes of search problem for which local knowledge can be used to derive an *optimal* search algorithm? Once more it may be possible to use a multi-dimension version of ANG to generate a new suite of test problems that exhibit far more chaotic behaviour than those that are currently used by the search community. It may be possible to analyse the relationship between known problems of relatively small complexity (as given by something like ANG) and see what sort of performance is possible from different styles of search algorithm.

A final area of research is more directly related to the results achieved in this thesis. One of the methods for analysing the results that we introduced was the probability price histograms for a time series model. This technique could be used for a much more complete analysis of the neural net model on the LGFC. For example, the same set of out-of-sample experiments could be run with the network training parameter being systematically adjusted. By analysing the price probability histograms it may be possible to find which features of the neural net solution were

most responsible for the high score ANTAS achieved. For example, it might be that the forecast horizon, in conjunction with the scaling of the training data, was crucial in achieving the good results. If this were the case, then less emphasis on the network architecture and training procedure might follow. Finding a systematic way in which to interpret a neural net output is of extreme importance. One of the most common complaints regarding the use of neural nets is the black-box quality of the results it achieves. In the context of the work contained within this thesis we have found a neural net that forecasts LGFC price trends with exceptional accuracy, and perturbing network inputs and training parameters might be one way in which to evaluate more deeply the aspects of the model responsible for the score. The price probability histograms could also be used to analyse methods for combining different forecasting models.

References

[AbuM93] Abu-Mostafa, Y., "Using Hints for Neural Network Training", Invited Talk at the *First International Workshop on Neural Networks in the Capital Markets*. London 1993.

[AcHS85] Ackley, D. H., Hinton, G. E., & Sejnowski, T. J., "A Learning Algorithm for Boltzman Machines." In *Cognitive Science*. Vol 9. pp. 147-169. 1985.

[AjSh91] Ajjanagadde, V., & Shastri. L., "Rules and Variables in Neural Networks." In *Neural Computation* Vol. 3:1. pp. 121-134. 1991.

[Alan94] Alander, J. T., "An Indexed Bibliography of Genetic Algorithms: Years 1957-1993," *Technical Report No 94-1*. Department of Information Technology and Production Economics, University of Vaasa. 1994.

[AnBi92] Anthony, M, & Biggs, N. L., *Computational Learning Theory: An Introduction*. Cambridge University Press. 1992.

[AnSm83] Angluin, D., & Smith, C.H., "Inductive Inference: Theory and Methods." In ACM Computing Surveys, pp 237-271 Vol 15 Number 3 Sept 1983.

[Anto89] Antonisse, J . "A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint." In *ICGA III*. San Mateo, CA, Morgan Kaufmann. 1989.

[Arei87] Areil, R., "Evidence on Intra-Month Seasonality in Stock Returns." In *Stock Market Anomalies*, Dimson, E. (Ed) Cambrige: Cambridge University Press, pp. 109-119 1987.

[BaBV93] Baestaens, D. E., van den Bergh, W. M., & Vaudrey, H., "Qualitive Credit Assessment Using a Neural Classifier." In *Proceedings of the First International Workshop of Neural Networks in the Capital Markets*. London. 1993.

[Back93] Bäck, T, "Optimal Mutation Rates in Genetic Search." In Forrest, S. (Ed), *Proceedings of the Fifth International Conference on Genetic Algorithms*. pp 2-9. Morgan Kaufmann. San Mateo, CA. 1993.

[BaHa89] Baum, E. B. & Haussler, D. "What size net gives valid generalization?" In *Neural Computation* Vol 1:1. pp. 151-160. 1989.

[BaHS91] Bäck, T., Hoffmeister, F., & Schwefel, H. P., "A Survey of Evolution Strategies", In Belew, R. K., & Booker, L. B. (Eds), *Proceedings of the Fourth International Conference on Genetic Algorithms*. pp 2-9. Morgan Kaufmann. San Mateo, CA. 1991.

[BaSc93] Bäck, T., & Schwefel, H. P., "An Overview of Evolutionary Algorithms for Parameter Optimization." In *Evolutionary Computation*. Vol 1:1. pp 1-24. 1993.

[Basu77] Basu, S., "Investment Perfromance of Common Stocks in Relation to their Price/Earnings Ratios: A Test of the Efficient Market Hypothesis." In *Journal of Finance*, 32 pp. 663-682 1977.

[Baue94] Bauer, R. J., *Genetic Algorithms and Investment Strategies*. John Wiley. New York. 1994.

[BEHW89] Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M., "Learning and the Vapnik-Chervonenkis Dimension." In *Journal of Association for Computer Machinery*. Vol 36:4. pp. 929-965. 1989.

[BeLe88] Becker, S, & Le Cun, Y., "Improving the Convergence of Back-Propagation Learning with Second Order Methods." In *Connectionist Models Summer School: 1988 Proceedings*. pp. 29-37. Morgan Kaufmann. San Mateo, CA. 1988.

[Beth81] Bethke, A. D., "Genetic Algorithms as Function Optimizers." *PhD Thesis, University of Michigan.* Dissertation Abstracts International, 41(9), 3503B (University Microfilms No. 8106101) 1980.

[BeMS90] Belew, R. K., McInerney, J., & Schraudolph, N. "Evolving Networks: Using the Genetic Algorithm with Connectionist Learning." *Technical Report* CS90-174. San Diego, University of California, Computer Science and Engineering Department 1990.

[BLLB91] Brock, W. Lakonishok, J. & LeBaron, B. "Simple Technical Trading Rules and the Stochastic Properties of Stock Returns." Technical Report No. 91-01-006. Sante Fe Institute New Mexico 1991.

[Binm77] Binmore, K. G., *Mathematical Analysis.* Cambridge University Press 1977.

[BrGo87] Bridges, C., & Goldberg, D. E., "An Analysis of Reproduction and Crossover in a Binary-Coded Genetic Algorithm." In *Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates. Hillsdale, NJ. 1987.

[CaSE89] Caruana, R. A, Schaffer, JD & Eschelman, L. J. "Using Multiple Representations to Improve Inductive Bias: Gray and Binary Coding for Genetic Algorithms." *Proc. 6th Intl. Workshop on Machine Learning.* Ithaca, NY, Morgan Kaufmann. 1989.

[Cast91] Casti, J., *Searching For Certainty: What Scientist Can Know About The Future.* Abacus Book USA William Morrow & Company Inc. 1991.

[Chat89] Chatfield C., *Analysis of Time Series: an Introduction.* 4th edition. Chapman & Hall. London. 1989.

[ChLi91] Chang, E. J., & Lippmann, R. P., "Using Genetic Algorithms to Improve Pattern Classification Performance." In *Advances in Neural Information Processing* 3, pp. 797-803, 1991.

[CHMR87] Coohen, J. P., Hedge S. U., Martin, W. N., & Richards, D. S., "Punctuated Equilibria: a Parallel Genetic Algorithm." In Grefenstette, J. J. (Ed), *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms and their Applications.* pp 148-154. Lawrence Erlbaum Associates. Hillsdale, NJ. 1987.

[CoGS88] Collins, E., Ghosh, S., & Scofield, C., "An Application of a Multiple Neural Network Learning System to Emulation of Mortgage Underwriting Judgements," In the *Proceedings of IEEE International Conference on Neural Networks.* Vol 2. pp. 459-466. San Diego. IEEE. 1988.

[Cohe82] Cohen, P.R., & Feigenbaum, E.A., (Eds) *The handbook of Artificial Intelligence Vol 3.* London Pitman 1982.

[Cybe89] Cybenko, G., "Approximation by superpositions of a sigmoidal function." In *Math. Control, Signals, and Systems.* Vol 2. pp. 303-314. 1989.

[Dav91] Davis, E. P., "Financial Disorder and the Theory of Crisis." In (Ed), Taylor, M. P., *Money and Financial Markets,* Blackwell Oxford 1991.

[Davi91] Davis, L., *Handbook of Genetic Algorithms,* Van Nostrand Reinhold. New York. 1991.

[David91] Davidor, Y., "A naturally occurring niche and species phenomenon: The model and first results." In Belew, R. K., & Booker, L. B. (Eds), *Proceedings of the Fourth International Conference on Genetic Algorithms.* pp 257-263. Morgan Kaufmann. San Mateo, CA. 1991.

[Debo94] Deboeck, G. J. (Ed), *Trading on the Edge.* John Wiley. New York. 1994.

[DEFH93] Der, R., Englisch, H., Funke, M., & Herrmann, M., "Prediction of Financial Time Series Using Hierarchical Self-Organized Feature Maps." In *Proceedings of the First International Workshop on Neural Networks in the Capital Markets.* London. 1993.

[DeJo75] De Jong, K. A. "An Analysis of the Behaviour of a Class of Genetic Adaptive Systems." *PhD Thesis Univeristy of Michigan.* Dissertations Abstracts International 36 (10) 5140B (University Microfilms No. 76-9381) 1975.

[DeKi94] Dekker, L., & Kingdon, J., "Development Needs for Diverse Genetic Algorithm Design." In Stender, J., Hillebrand, E., & Kingdon, J., (Eds) *Genetic Algorithms in Optimisation, Simulation and Modelling* (pp. 9-26). IOS Press. Amsterdam. 1994.

[Dick74] Dickenson, J.P., (Ed) *Portfolio Analysis: A Book of Readings.* Farnbourgh, Hants, Saxon House 1974.

[DLCD82] Dietterich T,G., London, R., Clarkson, K., & Dromey, R.. "Learning and Inductive inference". In Cohen, P., Fieigenbaum, E., (Eds) *The Handbook of Artificial Intelligence,* Kaufman, Los Altos, CA. pp 146-161.1982.

[Drey79] Dreyfus, H. L., *What Computers Can't Do.* Harper & Row. 1979.

[DuHa73] Duda, R., & Hart, P., *Pattern Classification and Scene Analysis.* Wiley Interscience Publication John Wiley & Sons New York 1973.

[DuSh88] Dutta, S , & Shekhar, S., "Bond Rating: A non-conservative application of neural networks," In the *Proceedings of IEEE International Conference on Neural Networks.* Vol 2. pp. 443-450. San Diego. IEEE. 1988.

[EaMa93] East, I. R., & Macfarlane, D., "Implementation in Occam of Parallel Genetic Algorithms on Transputer Networks", In Stender, J. (Ed), *Parallel Genetic Algorithms: Theory and Applications.* pp 43-63. IOS Press. Amsterdam. 1993.

[EsSc92] Eshelman, L. J., & Schaffer, D. J., "Real-coded genetic algorithms and interval schemata", In Whitley, D. (Ed) *Foundations of Genetic Algorithms 2.* Morgan Kaufmann. San Mateo, CA. 1992.

[FaDi88] Fama, E., & French, K.R., "Dividend Yields and Expected Stock Returns." In *Journal of Financial Economics,* Vol 22 pp 3-25 1988.

[Fama70] Fama, E., "Efficient Capital Markets: A Review of Theory and Empirical Work." In *Journal of Finance.* Vol. 25 (May). pp. 383-417. 1970.

[FaSi88] Farmer, J. D., & Sidorwich, J. J., "Can New Approaches to Nonlinear Modeling Improve Economic Forecasting? The Economy as an Evloving Complex System," *SFI Studies in the Science of Complexity,* Addison Wesley Publishing Company 1988.

[FeKi95] Feldman, K., & Kingdon, J., "Neural Networks and some applications in Finance." In *Journal of Applied and Mathematical Finance,* Chapman and Hall Oxford 1, 1995.

[FoAt90] Fogel, D. B., & Atmar, J. W., "Comparing Genetic Operators with Gaussian Mutations in Simulated Evolutionary Processes using Linear Systems." *Biological Cybernetics.* Vol 63:2. pp11-114. 1990.

[FCKL90] Feldman, J., Cooper, L., Koch, C., Lippman, R., Rumelhart, D., Sabbah, D., & Waltz, D., "Connectionist Systems." *Annual Review Computer Science.* Vol 4. pp. 369-381. Annual Review. 1990.

[FoCU91] Foster, B. F., Collopy, F., & Ungar, L., "Neural Network Forecasting of Short Noisy Time Series." Presented at the *ORSA TIMS National Meeting* May 1991.

[FoMi92] Forrest, S., & Mitchell, M., "Relative building-block fitness and the building-block hypothesis", In Whitley, D. (Ed) *Foundations of Genetic Algorithms 2.* pp 109-126. Morgan Kaufmann. San Mateo, CA. 1992.

[FoOW66] Fogel, L. J., Owens, A. J., & Walsh, M. J., *Artifical intelligence through simulated evolution.* John Wiley. New York. 1966.

[Frei91] Friedman, J. H. "Multivariate Adaptive Regression Splines." In *Annals of Statistics* 19 1-141 1991.

[Fren80] French, K.R., "Stock Returns and the Weekend Effect." In *Journal of Financial Economics* pp 55-69 Vol 8 1980.

[Funa89] Funahashi, K., "On the Approximate Realisation of Continuous Mappings by Neural Networks." In *Neural Networks.* Vol 2 , 183-192 1989.

[GaSu84] Gabr, M. M. & Subba Rao, T. "An Introduction to Bispectral Analysis and Bilinear Time Series Models." *Lecture Notes in Statistics.* Vol 24. New York Springer 1984.

[Gold89] Goldberg, D. E., *Genetic Algorithms In Search, Optimisation and Machine Learning.* Addison-Wesley. Reading, Mass. 1989.

[GoDC92] Goldberg, D. E., Deb, K., & Clark, J. H., "Accounting for Noise in the Sizing of Populations," In Whitley, D. (Ed) *Foundations of Genetic Algorithms 2.* pp. 127-140. Morgan Kaufmann. San Mateo, CA. 1992.

[GoDe91] Goldberg, D. E., & Deb, K., "A Comparative Analysis of Selection Schemes used in Genetic Algorithms." In Rawlins, G. (Ed), *Foundations of Genetic Algorithms.* pp69-93. Morgan Kaufmann. San Mateo, CA. 1991.

[GoDK91] Goldberg, D. E., Deb, K., & Korb, B., "Don't worry, be messy." In Belew, R. K., & Booker, L. B. (Eds), *Proceedings of the Fourth International Conference on Genetic Algorithms.* pp 24-30. Morgan Kaufmann. San Mateo, CA. 1991.

[GoFe94] Goonatilake, S., & Feldman, K., "Genetic Rule Induction for Financial Decision Making." In J. Stender, E. Hillebrand, & J. Kingdon (Eds) *Genetic Algorithms in Optimisation, Simulation and Modelling* (pp. 185-201). IOS Press. Amsterdam. 1994.

[GoKh95] Goonatilake, S., & Khebbal, S. (Eds), *Intelligent Hybrid Systems.* John Wiley. Chitchester. 1995.

[Grua93] Gruau, F., "Genetic Synthesis of Modular Neural Networks." The *fifth Annual Conference on Genetic Algorithms* 1993.

[Gros82] Grossberg, S., *Studies of Mind and Brain.* D Reidal. Dordrecht, Holland. 1982.

[GoWh93] Gordon, J., & Whitley, D., "Serial and parallel genetic algorithms as function optimizers", In Forrest, S. (Ed), *Proceedings of the Fifth International Conference on Genetic Algorithms.* pp 177-183. Morgan Kaufmann. San Mateo, CA. 1993.

[Gref84] Grefenstette, J. J., "GENESIS: A System for using Genetic Search Procedures." In *Proceedings of the 1984 Conference on Intelligent Systems and Machines.* pp. 161-165. 1984.

[Gref92] Grefenstette, J. J., "Deception Considered Harmful." In Whitley, D. (Ed), *Foundations of Genetic Algorithms 2.* pp 75-92. Morgan Kaufmann. San Mateo, CA. 1992.

[Grua95] Gruau, F., "Genetic Programming of Neural Networks: Theory and Practice." In Goonatilake, S., & Khebbal, S. (Eds), *Intelligent Hybrid Systems.* John Wiley. Chitchester. 1995.

[dGrW90] de Groot, C., & Wurtz, D., "Analysis of univariate time series with connectionist networks: A case study of two classical examples." Invited Talk at the *Munotec Workshop Neural Networks for Statistical and Economic Data.* Dublin. 1990.

[GrWh93] Gruau, F., & Whitely, D. "Adding Learning to the Cellular Development Process; A Comparative Study." In *Evolutionary Computing* Vol 1 No. 3 1993.

[GrWh93b] Gruau, F., & Whitely, D. "The Cellular Development of Neural Networks: the Interaction of Learning and Evolution." In Technical Report 1993.

[Hamm86] Hamming, R.W., *Coding and Information Theory.* Prentice Hall, Englewood Cliffs, New Jersey 1986.

[HaSa91] Harp, S. A., & Samad, T. "Genetic Synthesis of Neural Network Architecture," In Davis, L. (Ed) *Handbook of Genetic Algorithms.* Van Nostrand Reinhold. New York. 1991.

[HaSa91b] Harp, S. A., & Samad, T. "Genetic Optimization of Self-Organizing Feature Maps." In *Proceedings of International Joint Conference on Neural Networks.* Vol 1. pp. 341-345. Seattle, WA. IEEE. 1991.

[HaSW92] Hassibi, B., Stork, D., & Wolff, G. "Optimal Brain Surgeon and General Network Pruning." *Technical report* 9325, RIOCH California Research Center, Menlo Pk, CA. 1992.

[Haus92] Haussler, D., "Decision Theoretic Generalizations of the PAC Model for Neural Net and other Learning Applications." *Information and Computation.* Vol 100. pp. 78-150. 1992.

[Hebb49] Hebb, D. O., *Organisation of Behaviour, A Neuropsychological Theory.* New York: Science Editions. 1949.

[HeKP91] Hertz, J., Krogh, A., & Palmer, R. G., *Introduction to the Theory of Neural Computation.* Addison-Wesley. Redwood City. 1991.

[HOCR92] Hill, T., O'Conner, A. M., Marquez, L. & Remus, A. W., "Neural Network Models for Forecasting: A Review." In *Proceedings of the 25th Hawaii International Conference on Systems Sciences.* Vol 4 1992.

[Hint87] Hinton, G., "Conectionist Learning Proceedures," In *Carnegie Mellon Technical Report* CMU-CS-87-115 1987.

[Hirs87] Hirsch, Y., *Don't Sell Your Stocks on a Monday.* New York: Penguin 1987.

[HoGo94] Horn, J. & Goldberg, D., "GA Difficulty and the Modality of Fitness Landscapes". *IlliGAL Report No. 94006* 1994.

[Holl75] Holland, J. H., *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press. 1975.

[Holl86] Holland, J. H., "Escaping Brittleness." In Michalski, R., Carbonell, J., & Mitchell, T. (Eds) Machine Learning. Morgan Kaufmann, Vol 2 , 1986.

[Hopf82] Hopfield, J. J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," In *Proceedings of National Academic Science* (USA). Vol. 79. pp. 2554-2558. 1982.

[Horn91] Hornik, K. "Approximation Capabilities of Multilayer FeedForward Networks." In *Neural Networks.* Vol 4. pp. 251-257. Pergamon Press. 1991.

[HoSW89] Hornik, K., Stinchcombe, M. & White, H. "Multilayer FeedForward Networks Are Universal Approximators." In *Neural Networks.* Vol 2. pp. 359-366. 1989.

[Hugh90] Hughes, M., "Improving Products and Processes." In *Industrial Management and Data Systems.* Vol 6. pp 22-25. 1990.

[HWBG94] Haasdijk, E. W., et al, "Genetic Algorithms in Business", In Stender, J., Hillebrand, E., & Kingdon, J., (Eds) *Genetic Algorithms in Optimisation, Simulation and Modelling* (pp. 157-184). IOS Press. 1994.

[Inde93] The Independent, Business and City Page 3/8/93.

[Jone95] Jones, T., "A Model of Landscapes". *Santa Fe Inst. Tec Rep* 1995.

[Jaco88] Jacobs, R. A., "Increased Rates of Convergence through Learning Rate Adaptation." In *Neural Networks*. Vol. 1. pp.295-307. 1988.

[Jaff74] Jaffe, J.F., "Special Information and Insider Trading." In the *Journal of Business* pp 410-428 Vol 47 1974.

[Jega90] Jegadeesh, N., "Evidence of Predictable Behaviour of Securities Returns." In *Journal of Finance*, Vol 3 Number 45 pp. 881-898. 1990.

[Judd90] Judd, J. S., *Neural Network Design and the Complexity of Learning*. MIT Press. Cambridge, MA. 1990.

[KAYT90] Kimoto, T., Asakawa, K., Yoda, M., & Takeoda, M., "Stock Market Prediction System with Modular Neural Networks." In the *Proceedings of the International Joint Conference on Neural Networks*. Vol. 1,1-6. San Diego. 1990.

[Kier94] Kieran, V., "Growing Money From Algorithms," New Scientist No1954 Dec 94.

[KiDe95] Kingdon, J., & Dekker, L., "The Shape of Space", *Technical Report No. RN/95/23.* Dept. Computer Science. University College London. 1995.

[KiFe94] Kingdon, J., & Feldman, K. "Redundancy in Neural Nets: An Architecture Selection Procedure using In-Sample Performance." University College London. *Technical Report.* 1994.

[KiFe95] Kingdon, J., & Feldman, K., "Genetic Algorithms and some Applications in Finance." In *Journal of Applied and Mathematical Finance*, Chapman and Hall Oxford 1, 1995.

[King93] Kingdon, J., "Neural Nets for Time Series Forecasting: Criteria for Performance with an Application in Gilt Futures Pricing." In the *Proceedings of the First International Workshop on Neural Networks in the Capital Markets*. London. 1993.

[Knig21] Knight, F. H., Risk, *Uncertainty and Profit*, Boston: No. 16 in series of reprints of scarce texts in economics, London School of Economics 1921.

[Koho82] Kohonen, T., "Self Organized Formation of Topologically Correct Feature Maps." In *Biological Cybernetics*. Vol. 43. p 59. 1982.

[Koho89] Kohonen, T., *Self-Organization and Associative Memory*. 3rd Ed. Springer-Verlag. Berlin. 1989.

[Kosk92] Kosko, B., *Neural Networks and Fuzzy Systems*. Prentice Hall, Englewood Cliffs, NJ. 1992.

[Koza94] Koza, J., *Genetic Programming*. MIT Press 1994.

[LaFa87] Lapedes, A. S., & Farber, R., "Non-linear Signal Processing using Neural Networks: Prediction and System Modeling." *Technical Report* LA-UR-87 Los Alamos National Laboratory. 1987.

[LeCu89] Le Cun, Y., "Generalisation and Network Design Strategies." University of Toronto Technical Report CRG-TR-89-4, 1989.

[LeDS90] Le Cun, Y., Denker, J. S. & Solla, S. A.. "Optimal brain damage," In *Proceedings of Neural Information Processing* 2. pp. 598-605. Morgan Kaufmann. San Mateo, CA. 1990.

[LeLM94] Levin, A. U., Leen, T. K., & Moody, J. E. "Fast pruning using principle components." In *Proceedings of Neural Information Processing 6.* NIPS 6. Morgan Kaufmann. San Mateo, CA. 1994.

[LeVo90] Leipins, G & Vose, M., "Representation Issues in Genetic Algorithms". *Journal of Exp. and Theo. Art. Intel*, 2:101-115 1990.

[Lint65] Lintner. J., "The Valuation of Risk Assets and the Selection of Risky Investments in Stock Portfolios and Capital Budgets." In *Review of Economics and Statistics* Feburary 1965.

[LiVo91] Liepins, G. E., & Vose, M. D., "Deceptiveness and Genetic Algorithm Dynamics." In Rawlins, G. (Ed), *Foundations of Genetic Algorithms*. pp. 36-52. Morgan Kaufmann. San Mateo, CA. 1991.

[LoCo90] Lo A., & MacKinlay, C.A., "When are Contrarian Profits due to Stock Market Overreaction?" In *Review of Financial Studies*, Number 2 pp. 175-206. 1990.

[Lucu76] Lucus, R,E., "Econometric Policy Evaluation." In Brunner, K. and Meltzer, A. H., (Eds) *The Phillips Curve and Labour Markets, Carnegie-Rochester Conference Series on Public Policy*. Vol 6 Amsterdam: North Holland 1976.

[Malk90] Malkiel., B.G., *A Random Walk Down Wall Street*. W.W Norton Company 1990.

[Makr82] Makridakis, S., Anderson, A., Carbone, R., Fildes, R., Hilbon, M., Lewandowski, R., Netwon, J., Parzen, E., & Winkler, R., "The Accuracy of Extraporlation Methods: Results of a Forecastıng Competition." In the *Journal of Forecasting* 1,111,153 1982.

[MacK92] MacKay, D. J. C., "A practical Bayesian Framework for Backpropagation Networks." In *Neural Networks*. Vol 4. pp.448-472. 1992.

[MaGo92] Mahfoud, S. W., & Goldberg, D. E., "Parallel Recombinative Simulated Annealing: A Genetic Algorithm. *IlliGAL Report No. 92002* 1992.

[Mark59] Markowitz, J. *Portfolio Selection: Efficient Diversification of Investments*. John Wiley & Sons, New York 1959.

[MaSa94] Malliaris, M., & Salchenberger, L., "Neural Networks for Predicting Options Volatility." In *Proceedings of World Congress on Neural Networks*. Vol 2, pp. 290-295. San Diego. Lawrence Erlbaum Associates. Hillsdale, NJ. 1994.

[MaSe93] Martín-del-Brío, B. & Serrano-Cinca, C. "Self-organizing Neural Networks for the Analysis and Representation: Some Financial Cases." In *Neural Computation & Applications*. Vol 1, pp. 193-206. Spinger-Verlag. London. 1993.

[Mast93] Masters, T., *Practical Neural Network Recipes in C++*, Academic Press, San Diego, CA 1993.

[MaWh93] Mathias, K. & Whitley, D., "Remapping Hyperspace During Genetic Search: Canonical Delta Folding". *FOGA II* San Mateo CA, Morgan Kaufman. 167:186 1993.

[McPi43] McCulloch, W. W., & Pitts, W., "A Logical Calculus of Ideas Imminent in Nervous Activity." In *Bulletin of Mathematical Biophysics*. 5(115). pp. 115-133. 1943.

[Melt82] Meltzer A.,H., "Rational Expectations, Risk, Uncertainty and Market Responses." In (Ed.) Wachtel, P. *Crisis in the Economic and Financial Structure*, Solomon Bros. Center Series on Financial Institutions and Markets, Lexington MA: Lexington Books 1982.

[Meul92] Meulbroek L., K., "An Empirical Analysis of Illegal Insider Trading." In *Journal of Finance*. Vol XLVII, No 5 Dec. 1992.

[MeNe89] Mezard, M., & Nedal, J., "Learning in a Feedforward Layered Network." In *Journal of Physics*. Vol 22. 1989.

[MeSW81] Mendenhall, W., Scheaffer, R. L., & Wackerly, D. D., *Mathematical Statistics with Applications*. Duxbury Press Boston Massachusetts 1981.

[Mich92] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolutionary Programs*. Springer-Verlag. Berlin. 1992.

[MiCM86] Michalski, R.S., Carbonell, J.G., & Mitchell, T.M., (Eds) *Machine learning:an artificial intelligence approach.* - Vol.2. Los Altos, Ca. Morgan Kaufman, 1986.

[MiTH89] Miller, G. F., Todd, P. M., & Hedge, S. U. "Designing Neural Networks using Genetic Algorithms." In the *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 379-384. 1989.

[MiPa69] Minsky, M., & Papert, S., *Perceptrons.* Cambridge MA: MIT Press. 1969.

[MoDa89] Montana, D. J. & Davis, L., "Training Feed-Forward Neural Networks Using Genetic Algorithms." In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 746-767, 1989.

[Mood92] Moody, J. E. "The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems," In. Moody, J. E et al, (Eds.), *Advances in Neural Information Processing Systems.* Vol 4. pp. 847-854. San Mateo, CA: Morgan Kaufmann. 1992.

[Moss66] Mossin. J., "Equilibrium in a Capital Asset Market." In *Econometrica* Oct. 1966.

[MoSm89] Mozer, M. C. & Smolensky, P. "Skeletonization: A technique for trimming the fat from a network via relevance assessment." In Touretzky, D. S.(Ed.), *Advances in Neural Information Processing Systems.* Vol 1. pp. 107-115. San Mateo, CA: Morgan Kaufmann. 1989.

[MoUt92] Moody, J. E., & Utans, J., "Principled Architecture Selection for Neural Networks: Application to Corporate Bond Rating." In *Advances in Neural Information Processing Systems.* Vol. 4. 1992.

[Muhl91a] Mühlenbein, H, "Parallel Genetic Algorithms and Neural Networks as Learning Machines." In Evans, D. J., Joubert, G. R., and Liddell, H. (Eds), *Proceedings of the International Conference on Parallel Computing '91.* pp. 91-103. North-Holland, Amsterdam. 1991.

[Muhl91b] Mühlenbein, H., "Evolution in time and space - the Parallel Genetic Algorithm." In Rawlins, G. (Ed), *Foundations of Genetic Algorithms.* pp. 316-337. Morgan Kaufmann. San Mateo, CA. 1991.

[Muhl92] Mühlenbein, H., "How genetic algorithms really work: Mutation and Hill-Climbing." In Männer, R., & Manderick B. (Eds), *Parallel Problem Solving from Nature 2.* pp 15-26. Elsevier Science. Amsterdam. 1992.

[MuSB91] Mühlenbein, H., Schormisch, M., & Born, J., "The Parallel Genetic Algorithm as Function Optimizer." In Belew, R. K., & Booker, L. B. (Eds), *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann. pp. 271-278. San Mateo, CA. 1991.

[MuSV92a] Mühlenbein, H., & Schlierkamp-Voosen, D., "Predictive models for the breeder genetic algorithm I. Continuos Parameter Optimization." *Technical Report 92-121.* GMD, Germany. 1992.

[MuSV92b] Mühlenbein, H., & Schlierkamp-Voosen, D., "The distributed breeder genetic algorithm III. Migration." *Technical Report 92-122.* GMD, Germany. 1992.

[NaKr93] Nauck, D., & Kruse, R. "A Fuzzy Neural Network Learning Fuzzy Control Rules and Membership Functions by Fuzzy Error Backpropagation," In *Proceedings of IEEE International Conference on Neural Networks.* pp1022-1027. San Francisco. IEEE. 1993.

[Neal92] Neal, R. N., "Bayesian Training of Backpropagation Networks by the Hybrid Monte Carlo Method." *Technical Report* CRG-PR-92-1. University of Torronto 1992.

[Neda89] Nedal, J., "Study of a Growth Algorithm for Neural Networks." In *International Journal of Neural Systems* 1989.

[Nobl90] Noble, A., "Using Genetic Algorithms in Financial Services." In *Proceedings of the Two Day Conference on Forecasting and Optimisation in Financial Services.* IBC Technical Services. London. 1990.

[Norm88] Norman, M., "A Genetic Approach to Topology Optimisation for Multiprocessor Architectures," *Technical Report.* University of Edinburgh 1988.

[OdSh90] Odom, M. D., & Sharda, R., "A Neural Network Model For Bankruptcy Prediction." In *Proceedings of International Joint Conference on Neural Networks.* Vol 2. pp. 163-168. San Diego, CA. IEEE. 1990.

[OTW91] Ormerod, P., Taylor, J.C., Walker, T., "Neural Networks In Economics." In (Ed.) Taylor, M. P., *Money and Financial Markets.* Blackwell Oxford 1991.

[Pack90] Packard, N. H., "A Genetic Learning Algorithm for the Analysis of Complex Data." In *Complex Systems* Vol. 4. pp 543-572 1990.

[Pap61] Papet, S. "Some Mathematical Models of Learning." In (Ed.) Cherry, C., *Proceedings of 4th London Symposium on Information Theory.* Academic Press, New York 1961.

[Park82] Parker, D. B. "Learning Logic." *Invention Report S81-64,* File 1, Office of Technology Licensing, Stanford University, Stanford CA. 1982.

[Park87] Parker, D. B., "Optimal Algorithms for Adaptive Networks: Second Order Backpropagation, Second Order Direct Propagation, and Second Order Hebbian Learning," In *Proceedings of IEEE International on Neural Networks.* Vol 2. pp. 593-600. IEEE. 1987.

[PeCr85] Pennant-Rea, R., Crook, C., *The Economist Economics.* Penguin 1985.

[Radc92] Radcliffe, N. J., "Genetic Set Recombination", In Whitley, D. (Ed) *Foundations of Genetic Algorithms 2.* pp 203-219. Morgan Kaufmann. San Mateo, CA. 1992.

[RaSu94] Radcliffe, N. J., & Surrey, P. D., "The reproductive plan language RPL2: Motivation, architecture and applications", In Stender J., Hillebrand E., and Kingdon J. (Eds) *Genetic Algorithms in Optimisation, Simulation and Modelling.* pp. 65-94. IOS Press. Amsterdam. 1994.

[RadSu95] Radcliffe, N. J., & Surry, P.D., "Fundamental Limitations on Search Algorithms". *Technical report* 1995.

[Reed93] Reed, R. "Pruning Algorithms — A Survey," IEEE Transactions on Neural Networks. Vol 4:5. pp. 740-747. IEEE. 1993.

[Refe94] Refenes, A. N., (Ed) *Neural Networks in the Capital Markets.* John Wiley. Chitchester. 1994.

[RiCo86] Rizki, M., & Conrad, M., "Computing the Theory of Evolution." *Physica D.* Vol 22. pp 83-99. 1986.

[Ridl93] Ridley, M., "Mathematics of Markets." *Economist Survey: Frontiers of Finance.* 9th October, 1993.

[Rose62] Rosenblatt, F., *Principles of Neurodynamics.* Spartan Books. New York. 1962.

[RuHW86] Rumelhart, D. E., Hinton, G. E., & Williams, R. J., "Learning Internal Representations by Error Propagation." In D. E. Rumelhart & J. L. McClelland (Eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Vol I: Foundations (pp. 318-362). Cambridge MA: MIT Press/Bradford Books. 1986.

[Rume88] Rumelhart, D. E., "Learning and Generalization." In *IEEE International Conference on Neural Networks.* San Diego, CA. 1988.

[SaAn91] Sartori, M. A., & Antsaklis, P. J., "A Simple Method to Derive Bounds on the Size and to Train Multilayer Neural Networks." In *IEEE Transactions on Neural Networks*. Vol 2:4. pp. 467-471. IEEE. 1991.

[Scae87] Shaefer, C. G., "The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique", *ICGA II*. Hillsdale, NJ. Lawrence Erlbaum 1987.

[ScBe90] Schraudolph, N., & Belew, R., "Dynamic Parameter Encoding for Genetic Algorithms". *CSE Technical Report #CS 90-175* 1990.

[ScEs91] Schaffer, J. D., & Eshelman, L. J., "On Crossover as an Evolutionarily Viable Strategy," In Belew, R. K., & Booker, L. B. (Eds), *Proceedings of the Fourth International Conference on Genetic Algorithms*. pp 61-68. Morgan Kaufmann. San Mateo, CA. 1991.

[Scho90] Schoneburg, E., "Stock Market Prediction using Neural Networks: A Project Report." In Neurocomputing 2. pp. 17-27. Elsevier Science. 1990.

[ScWE90] Schaffer, J. D., Whitely, D., & Eshelman, L. J.,"Using Genetic Search to Exploit the Emerging Behaviour of Neural Networks." In Forrest, S. (Ed.) *Emergent Computation* pp 102-112, 1990.

[SeMM93] Serrano-Cinca, C., Mar-Molinero, C., & Martin-Del-Brio, B., "Topology-Preserving Neural Architectures and Multidimensional Scaling for Multivariate Data Analysis." In Proceedings of the First International Workshop on Neural Networks in the Capital Markets. London. 1993.

[Shap92] Shapiro, S.C., (Ed) *Encyclopedia of Artificial Intelligence*. New York Wiley 1992.

[Shar64] Sharpe. W., "Capital Asset Prices: A Theory of Market Equilbrium." In *Journal of Finance*, September 1964.

[Shil87] Shiller, R.J., "The Volatility of Stock Prices." In *Science*. 235 pp. 33-37 1987.

[ShPa90] Sharda, R., & Patel, R. "Neural Networks as Forecasting Experts: An Empirical Test." In *Proceedings of the 1990 UCNN Meeting*, Vol 2, pp 491-494 1990.

[ShPa90b] Sharda, R., & Patel, R. "Connectionist Approach to Time Series Predicition," *Oklahoma State University Working Paper 90-26* 1990.

[Simo63] Simons, G., F., *Introduction to Topology and Modern Analysis*. McGraw-Hill Inetrnational Book Company 1963.

[Smit93] Smith, M., *Neural Networks for Statistical Modelling*. Van Nostrum Reinhold New York 1993.

[Sten91] Stender, J., (Ed), *Parallel Genetic Algorithms in Theory and Practice*. IOS Press Holland 1991.

[Sont93] Sontag, E. D., "Some Topics in Neural Networks and Control," Rutgers University *Technical Report No. LS93-02*. 1993.

[SSRP92] Synder, J., Sweat, J., Richardson, M., & Pattie, D., "Developing Neural Networks to Forecast Agricultural Commodity Prices." In *Proceedings Hawaii International Conference on Systems Sciences*. pp. 516-522. IEEE. 1992.

[SuSi90] Surkan, A. J., & Singleton, J. C., "Neural Networks for Bond Rating Improved by Multiple Hidden Layers," In *Proceedings of International Joint Conference on Neural Networks*. Vol 2. pp. 157-162. San Diego, CA. IEEE. 1990.

[Spea92] Spears, W. M., "Crossover or mutation?", In Whitley, D. (Ed) *Foundations of Genetic Algorithms 2*. pp 221-238. Morgan Kaufmann. San Mateo, CA. 1992.

[SpJo91] Spears, W. M., & De Jong, K. A., "An Analysis of Multi-Point Crossover", In Rawlins, G. (Ed), *Foundations of Genetic Algorithms*. Morgan Kaufmann. San Mateo, CA. 1991.

[Sysw91] Syswerda, G., "A Study of Reproduction in Generational and Steady-State Genetic Algorithms", In Rawlins, G. (Ed), *Foundations of Genetic Algorithms*. pp. 94-101. Morgan Kaufmann. San Mateo, CA. 1991.

[Sysw92] Syswerda, G., "Simulated Crossover in Genetic Algorithms." In Whitley, D. (Ed) *Foundations of Genetic Algorithms 2*. pp. 239-256. Morgan Kaufmann. San Mateo, CA. 1992.

[Tong90] Tong, H., *Non-Linear Time Series: A Dynamic Systems Approach*. Oxford: Oxford Univeristy Press 1990.

[TaKa92] Tanigawa, T., & Kamijo, K., "Stock Price Pattern Matching System." In *Proceedings of International Joint Conference on Neural Networks*. Vol 2. pp. 465-471. IEEE. 1992.

[Tayl93] Taylor, J. G., *The Promise of Neural Networks*. Perspectives in Neural Computing, Springer-Verlag, London 1993.

[TdAF90] Tang, Z., de Almeida, C., & Fishwick, P., "Time Series Forecasting using Neural Nets vs. Box Jenkins Methodology," Presented at *International Workshop on Neural Networks* Feb. 1990.

[VaCh71] Vapnik, V. N., Chervonenkis, A., "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities." In *Theory of Probability and its Applications* 1971.

[Vali84] Valiant, L. G., "Theory of the Learnable." In *Communications of the ACM*. Vol. 27:11. pp. 1134-1142. 1984.

[Vapn82] Vapnik, V. N., *Estimation of Dependencies Based on Empirical Data*. Springer. Berlin. 1982.

[WaOe89] Wasserman, P. D., & Oetzel, R., M., *NeuralSource*. Van Nostrand Reinhold. New York. 1989.

[Wass89] Wasserman, P. D., *Neural Computing: Theory and Practice*. Van Nostrand Reinhold. New York. 1990.

[Watr87] Watrous, R. L., "Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Non-Linear Optimization." In *Proceedings of IEEE International on Neural Networks*. Vol 2. pp. 619-628. IEEE. 1987.

[WeGe93] Weigend, A. S., & Gershenfeld, N. A. (Eds), *Time Series Prediction : Forecasting the Future and Understanding the Past*. Addison-Wesley. Reading, MA. 1993.

[WeHR90] Weigend, A. S., Huberman, B. A., & Rumelhart, D. E., "Predicting the Future: A Connectionist Approach," *Stanford PDP Research Group Technical Report, PDP-90-01*. 1990.

[Werb74] Werbos, P. Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences. *Ph.D. Thesis* Harvard University. 1974.

[WeRH91] Weigend, A. S., Rumelhart, D. E. & Huberman, B. A. "Generalization by weight-elimination with application to forecasting." In (Eds) Lippmann, R. P *Advances in Neural Information Processing Systems*. Vol 3. pp. 875-882. San Mateo, CA: Morgan Kaufmann. 1991.

[WeRH91b] Weigend, A. S., Rumelhart, D. E., & Huberman, B. A., "Generalization by Weight Elimination applied to Currency Exchange Rate Prediction." In *Proceedings of the International Joint Conference on Neural Networks*. Vol 1. pp. 837-841. Seattle. IEEE. 1991.

[WhBo90] Whitley, D. & Bogart, C. "The Evolution of Connectivity: Pruning Neural Networks using Genetic Algorithms." In *Proceedings of International Joint Conference on Neural Networks*, Vol 1, p134. Washington DC. 1990.

[WhHa89] Whitley, D., & Hanson, T., "Optimizing Neural Networks using faster, more accurate Genetic Search." In *Proceedings of the Third International Conference on Genetic Algorithms*. pp. 391-396. Morgan Kaufmann. San Mateo, CA. 1989.

[WhKa91] Whitley, D., & Kauth, J., "GENITOR: A Different Genetic Algorithm." In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*. pp. 118-130. Denver, CO. 1988.

[Whit88] White, H., "Economic Prediction using Neural Nets: The case of the IBM daily stock returns." In *Proceedings of IEEE International Conference on Neural Networks*. Vol. 2. pp. 451-458. IEEE. 1988.

[Whit89] Whitley, D., "The GENITOR Algorithm and Selection Pressure: Why rank-based allocation of reproductive trials is best." In Schaffer, J. D. (Ed), *Proceedings of the Third International Conference on Genetic Algorithms*. pp 116-121. Morgan Kaufmann. San Mateo, CA. 1989.

[Whit91] Whitley, D. "Fundamental Principles of Deception in Genetic Algorithms." In Rawlins, G. (Ed) *Foundations of Genetic Algorithms*. Morgan Kaufmann. San Mateo, CA. 1991.

[WhSB89] Whitley, D., Starkweather, T., & Bogart, C., "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity." *Technical Report CS-89-117*. Colorado State University. 1989.

[Wils94] Wilson, C. L., "Self-Organizing Neural Network System for Trading Common Stocks." In *Proceedings IEEE International Conference on Neural Networks*. Vol 6. pp. 3651-3654. Orlando, FL. IEEE. 1994.

[WoMa95] Wolpert, D., & Macready, W., "No Free Lunch Theorems for Search." *Technical Report SF1-TR-95-02-010*. Santa Fe Institute. 1995.

[Wolp93] Wolpert, D. H., "On Overfitting Avoidance as Bias." *Technical Report SFI TR 92-03-5001* Santa Fe Institute 1993.

[WoWa91] Wong, F. S., & Wang, P. Z., "A Stock Selection Strategy using Fuzzy Neural Networks." In *Neurocomputing*. Vol 2:5,6. pp. 233-242. Elsevier. 1991.

[YuMa93] Yuret, D & de la Maza, M., "Dynamic Hill-Climbing: Overcoming the Limitations of Optimization Techniques". *2nd Turkish Symp on Art. Intel. and Neural Networks*. 208:212 1993.

[ZhMu93] Zhang, B., & Mühlenbein, H., "Genetic Programming of Neural Nets Using Occam's Razor." The *fifth Annual Conference on Genetic Algorithms* 1993.

Appendix A: Test Functions

TF1 Counting Ones

For binary coded string of length n:

$$f_1(\vec{x}) = \sum_{i=1}^{n} 1 \text{ if x=1, 0 otherwise}$$

F2 Royal Road 1

s_1 = 11111111**********; $c_1 = 8$
s_2 = ********11111111**********; $c_2 = 8$
s_3 = ****************11111111**********; $c_3 = 8$
s_4 = ************************11111111**********; $c_4 = 8$
s_5 = ********************************11111111**********; $c_5 = 8$
s_6 = **11111111**********; $c_6 = 8$
s_7 = **11111111********; $c_7 = 8$
s_8 = **11111111; $c_8 = 8$

$R_1(x) = \sum_i c_i \delta_i(x)$, where $\delta_i(x) = 1$ if $x \in s_i$, 0 otherwise.

TF3 Royal Road 2

s_1 = 11111111**********; $c_1 = 8$
s_2 = ********11111111**********; $c_2 = 8$
s_3 = ****************11111111**********; $c_3 = 8$
s_4 = ************************11111111**********; $c_4 = 8$
s_5 = ********************************11111111**********; $c_5 = 8$
s_6 = **11111111**********; $c_6 = 8$
s_7 = **11111111********; $c_7 = 8$
s_8 = **11111111; $c_8 = 8$
s_9 = 1111111111111111**********; $c_9 = 16$
s_{10} = ****************1111111111111111**********; $c_{10} = 16$
s_{11} = ********************************1111111111111111**********; $c_{11} = 16$
s_{12} = **1111111111111111; $c_{12} = 16$
s_{13} = 11111111111111111111111111111111**********; $c_{13} = 32$
s_{14} = ********************************11111111111111111111111111111111; $c_{14} = 32$

$R_2(x) = \sum_i c_i \delta_i(x)$, where $\delta_i(x) = 1$ if $x \in s_i$, 0 otherwise.

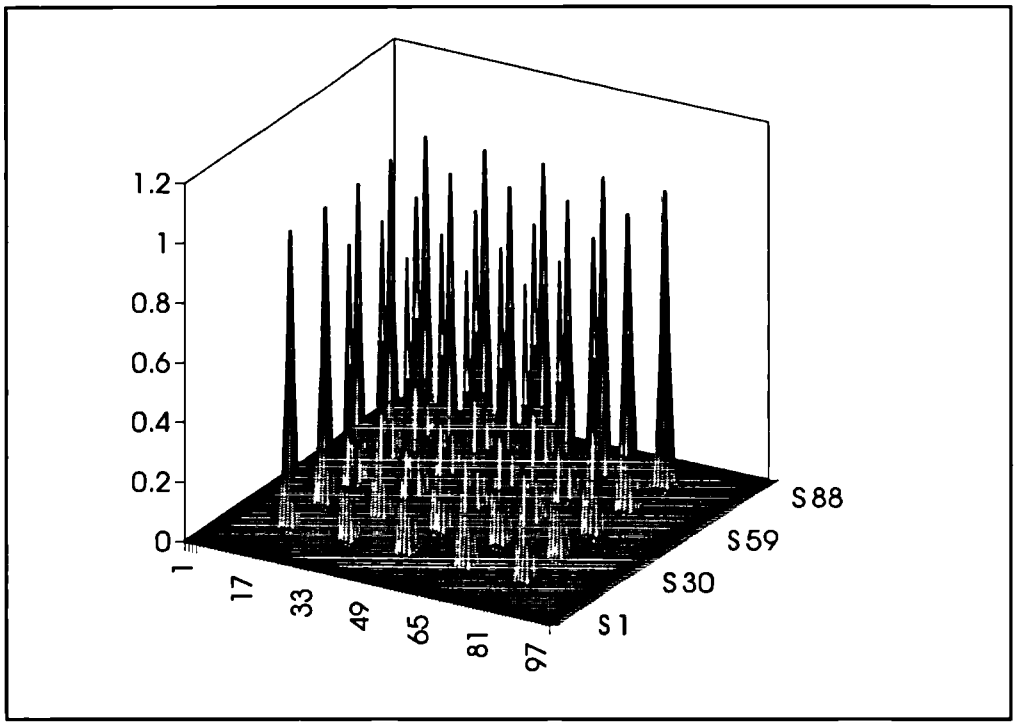
TF4 Shekel's Foxholes

Shekel's Foxholes in n dimensions:

$$a = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \dots & 32 & 32 & 32 \end{pmatrix}$$

$$K = 500$$
$$c_j = 1$$

$$f_4(\vec{x}) = 1/K + \sum_{j=1}^{25} \frac{1}{c_j + \sum_{i=1}^{n} (x_i - a_{ij})^6}$$

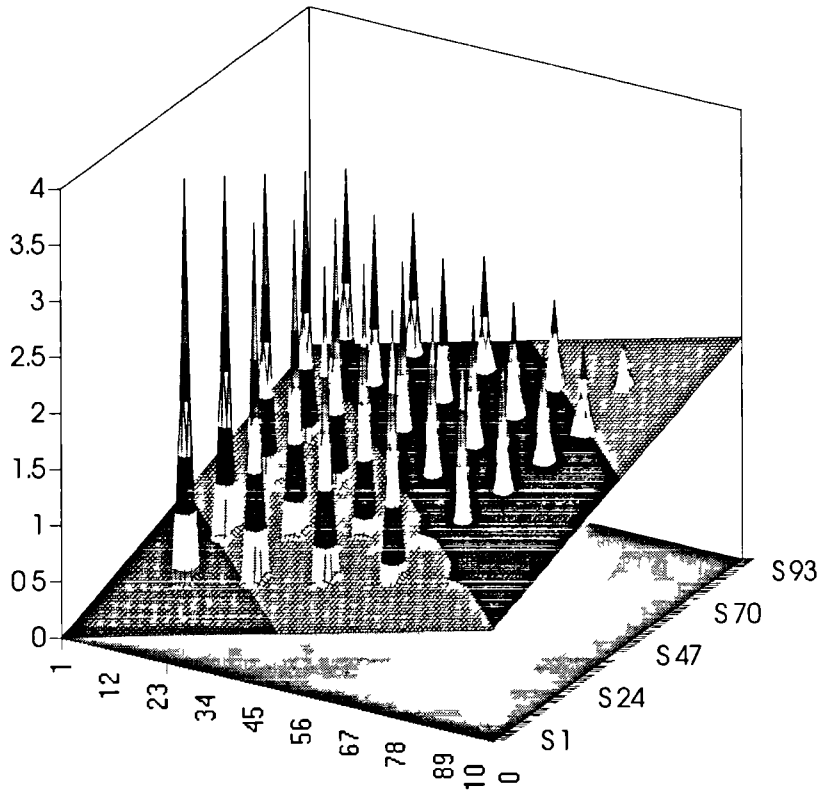


TF5 Shifted Shekel's Foxholes

Shifted Shekel's Foxholes in n dimensions:

$$p = 100$$
$$q = 2.5$$

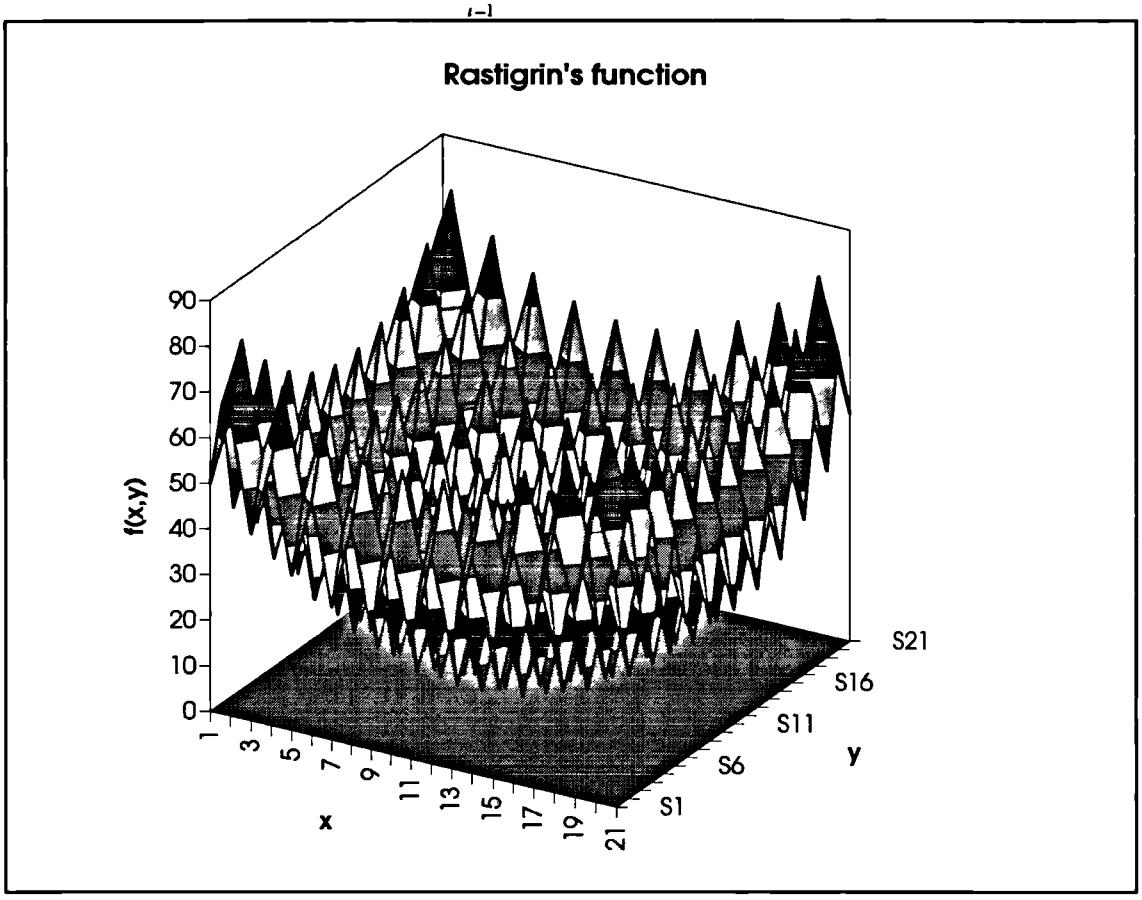
$$f_5(\vec{x}) = 2p + \left(\sum_{i=1}^{n} x_i \right) \left(\frac{1}{p} - q \, f_4(\vec{x}) \right)$$



TF6 Rastigrin's Function

Rastigrin's function in n dimensions:

$$f_6(\vec{x}) = \sum_{i=1}^{n} 10 + x_i^2 - 10\cos(2\Pi x_i)$$



TF7 4x 4-bit Multiple Deceptive

Chromosome 1		Chromosome 2		Chromosome 3		Chromosome 4	
Decoded value	Fitness +=	Decoded value	Fitness +=	Decoded value	Fitness +=	Decoded value	Fitness +=
0	8.0	0	18.0	0	6.0	0	2.0
1	4 0	1	26 0	1	30.0	1	8.0
2	22 0	2	6 0	2	18.0	2	10.0
3	10 0	3	8.0	3	4.0	3	26.0
4	0.0	4	2 0	4	16.0	4	30 0
5	30.0	5	16.0	5	2.0	5	6.0
6	12.0	6	30 0	6	26.0	6	4 0
7	2.0	7	0.0	7	14.0	7	12.0
8	20.0	8	12.0	8	12.0	8	14.0
9	14.0	9	28.0	9	0.0	9	22.0
10	28.0	10	20.0	10	24.0	10	20.0
11	26.0	11	22.0	11	10.0	11	28.0
12	16.0	12	14.0	12	22.0	12	0.0
13	6.0	13	24.0	13	8.0	13	16.0
14	24.0	14	4.0	14	28.0	14	18.0
15	18.0	15	10.0	15	20.0	15	24.0

DF1

3-bit GA deceptive problem (string length 3 in binary encoding).

Base 2	Base 10	Fitness
000	0	6
001	1	3
010	2	2
011	3	0
100	4	2
101	5	0
110	6	1
111	7	7

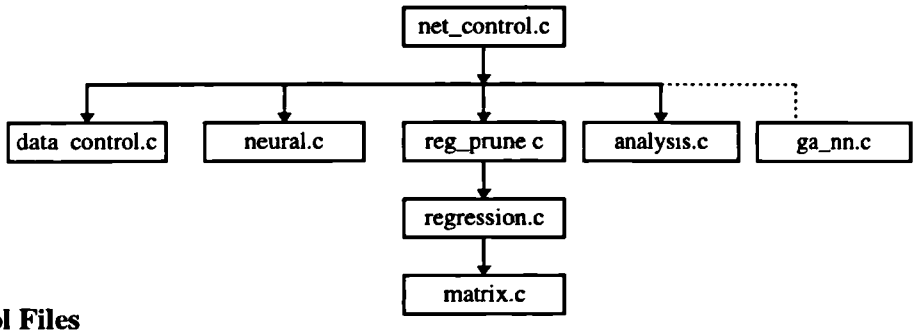
DF2

10 x 4-bit GA deceptive problem (string length 40 in binary encoding).

Base 2	Base 10	Fitness
0000	0	2.0
0001	1	8.0
0010	2	10.0
0011	3	26.0
0100	4	30.0
0101	5	6.0
0110	6	4.0
0111	7	12.0
1000	8	14.0
1001	9	22.0
1010	10	20.0
1011	11	28.0
1100	12	0.0
1101	13	16.0
1110	14	18.0
1111	15	24.0

Appendix B: ANTAS Outline Code

Neural Networks



Control Files
control_file
Headers:

---

neural_header.h

Description: *Contains the data structures for each of the neural network slaves. struct analysis_records, struct analysis_forecasts, struct Error_Profiles, struct data_files, struct control, struct node*

---

reg_prune.h

Description: *Contains the data structures for each of the regression pruning. struct net_data, struct nodes_history, struct candidate_node*

---

matrix.h

Description: *Contains the data structures for all matrix manipulations. struct matrix*

Source Files:

---

data_control.c

Description:
int read_data();
int moving_average();
int scale();

---

net_control.c

Description:

```
int read_control();          /*      read control file          */
int generate_control();
int get_data();
int load_control();          /*   loads up control struct       */
int unload_control();        /*   unloads values from comntrol struct   */
int check_reboot();
```

---

neural.c

Description:
double real_rand();
int set_layer();
int set_indexes();
int set_connections();

int set_activation_types();
int set_weights();
int set_up_net();
int get_num_nodes();
int get_activation();
double get_derivative();
int back_pass();
int weight_update();
int batch_weight_update();
int batch_bias_update();
int print_weights();
int print_connections();
int print_activations();
int print_netout();
int print_training_patterns();
int train_net();
int get_next_patterns();
int forecast();
int sign();
int file_output();
int file_errors();
int collect_recall_errors();
int validation_forecast_error();

---

reg_prune.c

Description:
*extern Matrix *allocate_matrix();*
extern check_reboot();
int read_connections();
int read_weights();
int read_input_vectors();
int read_output_vectors();
int activation_pass();
int act_history();
int set_layers();
int perform_regression();
int get_errors();
int copy_net_data();
int load_node_changes();
double recall_errors();
int keep_best_change();
int make_best_change();
int get_num_connections();
int check_nodes_connections();
int file_profiles();
int keep_record_of_connections();
int update_records();
int print_network_connections();
int fine_tune_effected_nodes();
int fixed_net_regression();
int fixed_load_node_changes();
int write_weights();
int write_connections();

int regression_reboot_file();
int set_up_from_reboot();
int last_connection();
int clean_out ();
int regression_pruning();

---

regression.c

Description:
*extern Matrix *allocate_matrix();*
extern int print_matrix();
extern int copy_matrix();
extern int delete_row();
extern int delete_column();
*extern Matrix *transpose_matrix();*
*extern Matrix *matrix_multiplication();*
extern double column_times_row();
extern double row_times_column();
extern reduce_column_zero();
extern reduce_row_zero();
extern scaler_mult_column();
extern scaler_mult_row();
extern scaler_mult_matrix();
extern minus_rows();
extern minus_columns();
extern add_columns();
extern add_rows();
*extern Matrix *identity_matrix();*
*extern Matrix *invert_matrix();*
extern swap_rows();
extern swap_columns();
extern row_swap_if_zero();
regression();

---

matrix.c

Description:
*Matrix *allocate_matrix();*
int free_matrix();
int print_matrix();
int copy_matrix();
int delete_row();
int delete_column();
*Matrix *transpose_matrix();*
*Matrix *matrix_multipl cation();*
double column_times_row();
double row_times_column();
reduce_column_zero();
reduce_row_zero();
scaler_mult_column();
scaler_mult_row();
scaler_mult_matrix();
minus_rows();
minus_columns();
add_columns();

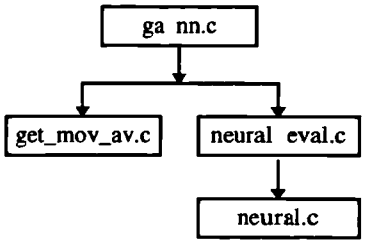
add_rows();
*Matrix *identity_matrix();*
*Matrix *invert_matrix();*
swap_rows();
swap_columns();
row_swap_if_zero();

---

analysis.c

Description:
extern int sign();
read_processed_files();
read_raw_files();
read_forecast_files();
rescale_forecast();
forecast_record_file();
make_analysis_file();
analysis(data,control_params)

Genetic Algorithm



Control Files
genetic_control
Headers:

genetic.h

Description: *Contains the data structures for each of the genetic algorithms. Struct indiviual, struct ga_control*

Source Files

ga_nn.c

Description:
int create_pop();
int print_pop();
int Int_Rand();
int decode_pop();
int trunk_selection();
double Flip();
int order();
int Holland_Cross();
int mutate();
int copy_individual();
int convert_to_base();
int find_chrom_length();
int load_from_reboot();
int read_ga_control();
int generate_ga_control();
int save_best();
int reboot_store();
extern Eval_Fitness();
extern check_reboot();

---

get_moving _average.c

Description:
int sign();
int mv_average();
int read_data();
get_moving_averages();

---

mav_tables.c

Description:
int sign();

int mv_average();
int read_data();

---

neural_evalutation.c

Description:

```
double  equation();
int read_control();     /*        read control file              */
int generate_control();
int load_control();     /*    loads up control struct            */
int unload_control(); /*    unloads values from comntrol struct    */
int store_fitness_cache();
int check_cache();
int first_call_set_ups();
int load_upreboot_fitness_cache();
int load_fitness_cache();
double read_NN_errors();
double read_forc_errors();
double read_in_error();
extern int create_pop();
extern int print_pop();
extern int Int_Rand();
extern int decode_pop();
extern int get_moving_averages();
extern double Flip();
Eval_Fitness();
```

Appendix C: ANTAS Results

ANTAS: Neural Network statistics.
Neural Net Id 7:.
Input Nodes 15.
Hidden Nodes 4.
Output Nodes 1.
Scale 0.500000
Moving Average 23
Training Patterns 250
Start Pattern 1000
Date - 9/3/95
Series: LGFC

key	fmov	tmov	rmov	bfmov	lfmov	hit
1000	-2.35	-3.46	-1.56	-0.88	0.18	1
1001	-7.01	-3.39	-2.75	-0.56	-2.65	1
1002	-0.77	-3.26	-2.69	-0.31	0.68	1
1003	-0.97	-3.19	-1.84	-1.22	0.5	1
1004	0.33	-3.13	-2.22	-1.19	1.23	0
1005	-0.92	-3	-1.03	-1.69	0.48	1
1006	0.79	-2.77	-3.38	-4.13	1.65	0
1007	0.27	-2.51	-2.06	-3.84	0.31	0
1008	3.99	-2.29	-1.59	-2.63	3.35	0
1009	0.62	-1 93	-1.38	-2.31	1.01	0
1010	1.78	-1.56	-2.28	-2 41	1.54	0
1011	2.88	-1.18	-2.69	-4.16	2.2	0
1012	2.74	-0.97	-3.28	-4.44	1.47	0
1013	-2.63	-0.62	-1.88	-5.19	-4.17	1
1014	-3.06	-0.29	-2.31	-4.16	-4.64	1
1015	-2.45	0.06	-2.94	-5.09	-3.86	1
1016	-4.27	0.33	-2.31	-3	-6.93	1
1017	4.69	0.55	0.16	-1.06	3.23	1
1018	-2.01	0.87	0.38	-0.81	-3.46	0
1019	-2.82	1.1	0.72	-0.44	-4.84	0
1020	4.01	1.29	2.22	1.44	3.2	1
1021	0.73	1.47	3.03	1.25	-0.7	1
1022	-2.78	1.75	4.59	1 91	-4 66	0
1023	-0.62	2.04	3.22	0.53	-1.49	0
1024	0.85	2.38	5.38	2.56	0.54	1
1025	-1.41	2.71	4.88	0.81	-2.93	0
1026	2.34	3.04	6.09	1	1.97	1
1027	1.4	3.45	4.13	-1.47	0.95	1
1028	-1.37	3.79	3.88	-1.09	-2.64	0
1029	-0.49	4.01	4.16	-1.38	-1.84	0
1030	5.57	4.21	3.19	-2.03	5.04	1
1031	-0.11	4.45	2.66	-4.31	-1.09	0
1032	2.29	4.58	2.94	-2.47	1.63	1
1033	-0.19	4.62	4.09	-0.75	-1.31	0

key	fmov	tmov	rmov	bfmov	lfmov	hit
1034	0.77	4.54	4	0.44	-0.13	1
1035	4.96	4.54	4.38	-0.19	6.17	1
1036	-0.12	4.45	5.75	1.5	-0.95	0
1037	1.72	4.4	5.34	1.88	1.75	1
1038	5.25	4.26	6.47	3.69	5.92	1
1039	3.69	4.22	5.66	2.94	4.09	1
1040	-0.63	4.21	5	2.44	0.25	0
1041	3.39	4.12	5.16	3.94	4.13	1
1042	5.1	4.09	6.22	4.53	6.4	1
1043	3.02	3.99	5.09	4.84	4.42	1
1044	-1.87	3.99	4.06	1.88	-1.57	0
1045	2.01	3.91	2.81	1.97	2.98	1
1046	4.14	3.86	3.13	1.41	5.54	1
1047	-1.77	3.83	3.22	0.44	-1.5	0
1048	1.85	3.79	3.84	-0.97	2.88	1
1049	2.65	3.74	2.78	-2	3.29	1
1050	3.48	3.61	3.31	-1.09	4.42	1
1051	6.85	3.47	3.5	0.5	8.34	1
1052	1.72	3.33	2.13	-1.69	2.49	1
1053	4.14	3.2	2.56	-1.91	5.68	1
1054	3.78	3.01	0.44	-2.97	4.98	1
1055	2.83	2.85	2.91	-0.59	4.07	1
1056	2.1	2.7	2.09	-1.41	3.15	1
1057	2.21	2.61	3	-0.31	3.43	1
1058	2.52	2.42	3.59	1.28	3.82	1
1059	2.82	2.23	4.78	2.91	4.35	1
1060	1 54	2.02	4.25	3.13	2.35	1
1061	2.09	1.88	3.47	2.06	3.25	1
1062	2.35	1.67	2.56	2.06	4.05	1
1063	2.77	1.43	1.78	3.03	4.28	1
1064	0.11	1.23	2	3.94	0.78	1
1065	2.81	1.04	1.94	3.44	3.85	1
1066	2.27	0.98	1.28	2.66	3.32	1
1067	2.11	0.79	0.72	2.97	3.65	1
1068	1.7	0.62	0.66	2.03	3.14	1
1069	1.65	0.37	-1.09	1.88	2.54	0
1070	1 91	0.11	-1.16	1.16	3.29	0
1071	1.63	-0.22	-1.09	1.41	2 66	0
1072	0.94	-0.48	-0.47	1.31	1 87	0
1073	-2 06	-0.75	-1.5	1.56	-1.48	1
1074	1.05	-0.98	-2.03	1.69	2.46	0
1075	-1.41	-1.18	-2.44	0.25	-0.63	1
1076	0.38	-1.4	-1 84	0.88	1.89	0
1077	-0.19	-1.61	-0.84	0.69	1.6	1
1078	-0.83	-1.74	-1.44	-0.53	0.61	1
1079	-0.6	-1.86	-1.81	-0.06	1.09	1
1080	-1.17	-1.99	-2.91	-0.75	0.62	1
1081	-0 92	-2	-2.38	-0.31	0.33	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1082	-0.38	-2.01	-2.75	-0.78	0.5	1
1083	-1.52	-2.01	-1.66	-0.75	-0.74	1
1084	-0.52	-2.03	-2.72	-1.44	0.28	1
1085	-2.31	-2.04	-2.84	-1.03	-2.09	1
1086	-0.87	-2.07	-2.69	0.38	-0.56	1
1087	-0.55	-2.03	-3.16	0	-0.09	1
1088	-0.93	-2.04	-2.84	0.59	-0.75	1
1089	-2.02	-2.13	-1.63	0.94	-2.05	1
1090	-1.59	-2.19	-2.09	0.91	-1.56	1
1091	-0.15	-2.25	-2.31	0 88	0.13	1
1092	-0.04	-2.23	-1.41	0.19	0.29	1
1093	-1.48	-2.25	-1.44	0.72	-1.47	1
1094	0.66	-2.3	-1.09	1.25	1.28	0
1095	-0.84	-2.42	-0.94	1.28	-0.91	1
1096	-0.86	-2.52	-1.69	0.81	-0.89	1
1097	-0.37	-2.6	-2.66	-0.03	-0.25	1
1098	-1.94	-2.68	-1.59	-0.53	-2.12	1
1099	-0.91	-2.73	-2.09	-0.94	-1.01	1
1100	-0.13	-2.79	-2.81	-0.44	-0.04	1
1101	-0.11	-2.83	-2.97	-2.03	-0.1	1
1102	-2.59	-2.81	-3.13	-2.47	-3.02	1
1103	-0.77	-2.8	-2.5	-1.47	-1.02	1
1104	-2.82	-2.84	-2.81	-2.22	-3.32	1
1105	-2.44	-2.84	-3.81	-2.78	-3.05	1
1106	-1.21	-2.88	-4.53	-2.66	-1.79	1
1107	0.93	-2.9	-5	-3.41	0.69	0
1108	-0.59	-2.9	-4.59	-3.03	-1.17	1
1109	-3.33	-2.88	-4.44	-2.22	-4.35	1
1110	0.35	-2.91	-4.44	-1.88	-0.14	0
1111	-0.03	-2.89	-4.25	-2	-0.79	1
1112	-1.34	-2.81	-2.53	-0.78	-2.2	1
1113	-1.21	-2.73	-1.59	-0.5	-2.12	1
1114	-3.47	-2.67	-2.19	-0.34	-4.76	1
1115	0.74	-2.66	-2.25	0.53	-0.12	0
1116	0.27	-2.64	-1.31	0.81	-0.65	0
1117	-0.9	-2.53	-2.03	-0.44	-1.89	1
1118	-2.95	-2.39	-1.47	0.06	-4.69	1
1119	-1.98	-2.2	-1.66	-0.06	-3.14	1
1120	1.06	-2.05	-2.25	-0.25	0.36	0
1121	1.35	-1.94	-2.38	0.13	0.65	0
1122	0.57	-1.82	-1.53	0.34	-0.21	0
1123	0.12	-1.69	-0.94	0.88	-0.69	0
1124	-1.35	-1.59	-1.06	0	-2.55	1
1125	2.55	-1.54	-1.88	-0 44	2.39	0
1126	0.79	-1.47	-2.34	-1.31	0.46	0
1127	1.83	-1 43	-2.19	-0.72	1.48	0
1128	0.39	-1.41	-1.44	-0.34	0.1	0
1129	-0.95	-1.35	-1.25	-0.16	-1.51	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1130	1.83	-1.35	-0.59	0.66	1.87	0
1131	2.06	-1.38	-1 25	0.53	2.6	0
1132	-1.85	-1 36	-1.78	-0.19	-2.38	1
1133	-0.08	-1.36	-1.59	0.34	-0.23	1
1134	0.87	-1.38	-1.31	0.13	1.01	0
1135	0.73	-1.41	-0.28	0.41	1.65	0
1136	0.51	-1.5	-0.56	1.56	0.7	0
1137	1.25	-1.54	-0.47	1.34	1.74	0
1138	0.24	-1.5	-1.22	-0.25	0.45	0
1139	0.61	-1.48	-1	0.03	0.86	0
1140	0.59	-1.52	-0.63	0.06	1.42	0
1141	0.16	-1.55	-1.56	-0.91	0.29	0
1142	-1.22	-1.64	-2.28	-0.66	-1.38	1
1143	0.31	-1.66	-1.81	-1.47	0.4	0
1144	0.95	-1.62	-2.28	-2.31	1.24	0
1145	-1.19	-1.55	-2.06	-2.5	-1.59	1
1146	0.56	-1.51	-1.56	-1.97	0.49	0
1147	1.42	-1.51	-3.13	-3.38	1.47	0
1148	-0.28	-1.5	-2.88	-3.28	-0.82	1
1149	-0.33	-1.49	-1.34	-1.88	-0.83	1
1150	1.41	-1.44	-1.72	-1.59	1.2	0
1151	-1	-1.45	-2.34	-0.94	-1.93	1
1152	1	-1.44	-2.03	-1.16	0.72	0
1153	1.48	-1.42	-2.53	-1.66	1.19	0
1154	2.06	-1.36	-1.91	-0.28	2.03	0
1155	1.25	-1.32	-0.75	-0.16	0.86	0
1156	1.46	-1.23	-0.03	0.13	1.12	0
1157	2.03	-1.15	-0.25	-0.09	2.07	0
1158	2.07	-1.07	-0.28	-0.03	2.03	0
1159	1.97	-0.9	-0.38	0.19	1.62	0
1160	3.46	-0.76	-0.34	0.72	3.51	0
1161	1.14	-0.68	-0.06	0.81	0.51	0
1162	2.78	-0.58	-1.13	0.03	2.57	0
1163	2.3	-0.43	-0.53	-0.03	2	0
1164	2.61	-0.29	-0.94	0.44	2.46	0
1165	1.73	-0.14	-1.03	-0.06	1.38	0
1166	2.78	0	-0.81	0.16	2.85	0
1167	2.45	0.09	-0.22	0.28	2.73	0
1168	2.49	0.14	-0.16	1	2.9	0
1169	0.35	0.24	0.16	0.75	0.16	1
1170	2.12	0.37	0.75	0.72	2.6	1
1171	-2.38	0.47	0.28	0.28	-0.42	0
1172	1.94	0.54	0.5	0.5	2.56	1
1173	-0.23	0.59	0.69	0.78	1.21	0
1174	1.23	0.7	1.13	0.88	1.87	1
1175	0.74	0.8	1.06	0.56	1.79	1
1176	0.24	0.97	1.09	0.25	0.77	1
1177	0.22	1.15	1.19	0.09	1.37	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1178	0.95	1.29	1.41	0.47	1.62	1
1179	0.83	1.38	1.16	-0.31	1 67	1
1180	0.46	1.5	1.97	-0.5	0.9	1
1181	0.7	1.59	2.63	-0.22	1.28	1
1182	0.86	1.66	2	-0.34	1.75	1
1183	0.8	1.78	1.41	-0.63	1.34	1
1184	-0.23	1.88	0.88	-0.88	-0.14	0
1185	0.5	2	1.41	-1	1.34	1
1186	0.84	2.08	1.94	-1.06	1.56	1
1187	1.16	2.17	2.94	-0.38	1 75	1
1188	0.84	2.27	3.19	-0.69	1.48	1
1189	1.12	2.37	2.19	-1.66	1.46	1
1190	0.87	2.45	1.97	-1.38	1 55	1
1191	1.49	2.54	2.53	-0.91	1.68	1
1192	1.09	2.61	2.31	-1.56	1.1	1
1193	1.55	2.62	2.44	-0.78	1.69	1
1194	1.4	2.63	2.94	-0.47	1.49	1
1195	2.01	2.66	2.84	-0.66	2.28	1
1196	0.95	2.72	3.34	-0.53	1.79	1
1197	1.82	2.77	3.06	-0.78	1.97	1
1198	1.62	2.79	3	-0.88	1.58	1
1199	1.49	2.76	3.5	0.06	1.37	1
1200	2.08	2.73	3.38	0.28	2.3	1
1201	1.48	2.72	3.25	0.34	2.32	1
1202	2.05	2.75	3.28	-0.06	2.48	1
1203	1.92	2.73	3.66	0.59	2 21	1
1204	2.11	2.69	2.84	-0.22	2.21	1
1205	2.13	2.62	2.19	-0.06	2 3	1
1206	2.77	2.53	2.16	0	3.36	1
1207	2.33	2.46	2.31	0.41	2 69	1
1208	2.05	2.33	2.38	1.31	2.26	1
1209	1.78	2.21	2.47	1.34	2.06	1
1210	1.71	2.1	2.22	1.38	2.45	1
1211	2.27	1.96	2.5	1.84	2 99	1
1212	1.89	1.82	2.09	1.63	2.32	1
1213	1.91	1.66	2.53	2.44	2.77	1
1214	1.59	1.5	2.09	2.06	2.47	1
1215	1.62	1.33	1.44	2.16	2.46	1
1216	0.89	1.15	0.81	1.91	2 03	1
1217	0.92	0.97	0.75	2.31	2 09	1
1218	0.67	0.83	1.22	2.56	1.71	1
1219	0.27	0 66	0.44	2.59	1.5	1
1220	0.59	0.51	0.34	2	1 51	1
1221	-0.5	0.36	0.34	1.41	0 94	0
1222	-2.6	0.25	0.5	0.97	-0 69	0
1223	-0.9	0.12	-0.03	1.16	0 37	1
1224	-0.43	-0.01	-0.44	1.44	0 6	1
1225	-1.24	-0.21	-0.22	2.09	-0 04	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1226	-1.31	-0.41	-0.34	2.09	-0.16	1
1227	-1.2	-0.55	-1.19	1.25	-0.21	1
1228	-1.8	-0.66	-2	0.5	-0.79	1
1229	-1.51	-0.78	-1.16	0.06	-0.66	1
1230	-1.87	-0.92	-1.63	-0.53	-1.05	1
1231	-1.5	-1.02	-1.13	0.09	-0.59	1
1232	-1.12	-1.12	-0.88	0.91	-0.77	1
1233	-1 87	-1.23	-0.34	1.09	-1.37	1
1234	-0.75	-1.34	-0.47	0.94	-0.52	1
1235	-2.07	-1.42	-1	0.06	-1.71	1
1236	-2.48	-1.47	-2.03	-0.31	-2.17	1
1237	-2.35	-1.51	-2.38	-0.38	-2.1	1
1238	-1.33	-1.54	-1.94	-0.47	-1.19	1
1239	-1.97	-1.52	-1.66	-0.09	-1.92	1
1240	-1.63	-1.42	-2.03	-0.16	-1.71	1
1241	-1.7	-1.34	-1.94	-0.22	-1.82	1
1242	-1 63	-1.24	-1.81	-0.38	-1.92	1
1243	-1.07	-1.13	-2.06	-1.22	-1.41	1
1244	-0.95	-1.05	-2.06	-1.34	-1.29	1
1245	-0.6	-0.98	-2.03	-1.56	-0.93	1
1246	-0.75	-0.87	-1.88	-1.47	-1.15	1
1247	-0.51	-0.76	-1.56	-1.41	-0.95	1
1248	-0.8	-0.62	-1.31	-1.22	-1.3	1
1249	0	-0.46	-1.06	-0.59	-0.02	1
1250	-0.34	-0.29	-0.66	-0.81	-0.53	1
1251	-0.56	-0.12	0.25	-0.81	-1.11	0
1252	-0.33	0.07	0.81	-0.97	-0.5	0
1253	-0.36	0.31	0.69	-1.63	-0.91	0
1254	-0.44	0.55	1.31	-1.44	-0.93	0
1255	-0.45	0.79	0.91	-1.41	-0.92	0
1256	-1.05	1	1.38	-0.69	-1.49	0
1257	-0.16	1.21	2.06	-0.63	-0.87	0
1258	0.05	1.41	1.53	-0.78	-0.32	1
1259	0.04	1.6	1.28	-0.5	-0.49	1
1260	0.17	1.76	1.16	-0.16	-0.46	1
1261	0.7	1.93	1.97	-0.5	0.19	1
1262	0.56	2.08	2.41	-0.53	-0.1	1
1263	0.56	2.15	2.34	-0 25	0.12	1
1264	0.34	2.17	3.5	0.38	0.07	1
1265	0.16	2.22	3.63	-0.09	0.02	1
1266	0.73	2.29	3.53	-0.44	0.41	1
1267	0.17	2.34	2.75	0.19	0.1	1
1268	0.87	2.38	2.81	0.53	0.77	1
1269	0.06	2.37	2.78	0.53	0.07	1
1270	0.48	2.41	2.75	0.19	0.43	1
1271	0.48	2.47	2.47	0.13	0.49	1
1272	0.77	2.54	2.72	0.72	1.2	1
1273	0.55	2.56	2.75	0.88	0.82	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1274	0.7	2.57	1.91	0.28	0.91	1
1275	0.34	2.6	1.28	0.06	0.45	1
1276	0.54	2.57	1.88	0.5	0.66	1
1277	0.75	2.53	3	0.84	1.02	1
1278	1.03	2.54	1.91	-0.81	1.39	1
1279	0.42	2.58	2.34	-0.84	0.59	1
1280	0.36	2.6	1.84	-0.59	0.51	1
1281	0.36	2.58	2.5	-0.28	0.45	1
1282	0.64	2.56	2.53	-0.63	0.9	1
1283	0.53	2.55	2.81	-0.63	0.74	1
1284	0.55	2.54	2.47	-0.81	0.77	1
1285	0.36	2.5	2.66	0.16	0.5	1
1286	0.31	2.48	3	0.69	0.48	1
1287	0.51	2.5	2.72	0.41	0.7	1
1288	0.5	2.44	2.78	0.91	0.73	1
1289	0.51	2.29	3.88	2.13	0.72	1
1290	0.67	2.15	3.59	2.53	1.13	1
1291	0.65	2	3.25	2.13	1.17	1
1292	0.49	1.82	2.28	2.16	0.86	1
1293	0.26	1.63	2.25	1.63	0.4	1
1294	-0.04	1.42	2.28	1.84	0.29	0
1295	0.42	1.18	2.5	2.47	1.02	1
1296	0.32	0.97	1.88	1.69	0.63	1
1297	0.32	0.75	1.47	1.38	0.64	1
1298	0.11	0.5	1.59	1.63	0.34	1
1299	0.27	0.26	0.63	1.81	0.42	1
1300	-0.02	0.02	-0.53	1.34	0.36	1
1301	-0.33	-0.25	-1.22	0.56	0.09	1
1302	0.08	-0.51	-1.13	1.19	0.29	0
1303	0.1	-0.83	-2.28	0.88	0.25	0
1304	-0.03	-1.07	-1.91	1.22	0.22	1
1305	0.12	-1.31	-2.41	0.69	0.22	0
1306	-0.29	-1.53	-2.66	0.13	-0.13	1
1307	-0.59	-1.73	-2.34	0.47	-0.41	1
1308	-0.08	-1.9	-2.28	0.97	-0.04	1
1309	-0.53	-2.09	-2.84	1 16	-0.44	1
1310	0.05	-2.27	-2.81	0.25	-0.25	0
1311	-0.81	-2.39	-2.81	-0 19	-0.67	1
1312	-1.63	-2.43	-2.34	-0.69	-1 37	1
1313	-0.56	-2.4	-2.41	-1.84	-0.68	1
1314	-0.1	-2.4	-3.94	-1.84	-0.12	1
1315	-0.33	-2.4	-3.34	-0.97	-0.61	1
1316	-0.51	-2.41	-3.19	-0.66	-0.67	1
1317	-0.36	-2.43	-2.75	0.06	-0.5	1
1318	-0.75	-2.4	-2.09	-0.41	-0.97	1
1319	-0.13	-2.43	-2.22	-0.06	-0.32	1
1320	-0.33	-2.48	-2.78	0.19	-0.51	1
1321	-0.28	-2.52	-2.47	0.81	-0.35	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1322	-1.47	-2.54	-2.31	0.59	-1.63	1
1323	-1.16	-2.56	-1.25	1.03	-1.39	1
1324	-0.44	-2.61	-0.72	1.78	-0.55	1
1325	-0.34	-2.67	-1.13	1.34	-0.39	1
1326	-0.23	-2.65	-2.28	0.63	-0.4	1
1327	-0.45	-2.69	-2.16	1.16	-0.91	1
1328	-0.29	-2.73	-2.84	0.41	-0.4	1
1329	-0.65	-2.81	-1.84	0.81	-1.07	1
1330	-0.44	-2.87	-2.97	-0.5	-0.63	1
1331	-1.37	-2.98	-3.53	-0.91	-1.82	1
1332	-1.24	-3 06	-3.72	-1.16	-1.65	1
1333	-0.88	-3.19	-3.28	-0.78	-1.16	1
1334	-1.37	-3.29	-3.28	-0.63	-1.92	1
1335	-1 06	-3.41	-3.53	-0.84	-1.47	1
1336	-0.95	-3.54	-3.84	-0.72	-1.33	1
1337	-0.48	-3.66	-3.53	-1.25	-0.74	1
1338	-1.36	-3.76	-4.06	-2.28	-1.8	1
1339	-0.48	-3.85	-4.16	-2.28	-0.83	1
1340	-0.51	-3.92	-4.59	-2.25	-0.7	1
1341	-0.49	-3.95	-3.5	-2.41	-0.67	1
1342	-0.61	-3.93	-4.81	-2.53	-1.06	1
1343	-1	-3.83	-4.66	-2.84	-1.42	1
1344	-1.93	-3.76	-5.34	-2.69	-2.95	1
1345	-0.58	-3.71	-4.75	-2.53	-1.11	1
1346	-0.3	-3.68	-3.94	-2.38	-0.97	1
1347	-0.44	-3.6	-3.75	-2.81	-0.7	1
1348	-0.32	-3.54	-3.88	-1.63	-0.9	1
1349	-0.02	-3.42	-4.47	-0.94	-0.38	1
1350	-0.35	-3.29	-4.25	-0.56	-0.67	1
1351	-0.13	-3.2	-4.44	-0.09	-0.38	1
1352	-0.26	-3.02	-2.47	-0.78	-0.54	1
1353	0.27	-2 88	-2.66	-0.22	-0.71	0
1354	-0.11	-2 67	-1.13	-0.09	0	1
1355	-0.13	-2.49	-2.13	0.03	-0.06	1
1356	0.02	-2.33	-2.25	0.72	-0.33	0
1357	0 83	-2.2	-2.5	1.03	0.14	0
1358	0.19	-2.07	-1.75	1.22	-0.12	0
1359	0.88	-1 92	-2.34	0.59	0.26	0
1360	0 63	-1 77	-0.78	0.31	0.33	0
1361	0	-1.57	-1.06	0 41	-0.13	0
1362	0.16	-1.37	-2.19	-0.16	0.15	0
1363	0.05	-1.17	-0.38	0.97	-0.05	0
1364	0.52	-1.04	-0.44	0.09	0.52	0
1365	0.12	-0.9	0.03	0.38	-0.06	1
1366	0.95	-0.8	-0.5	-0.03	1.12	0
1367	0.38	-0.7	-1.59	-0.16	0.31	0
1368	0.23	-0.59	-1.78	-0.81	0.3	0
1369	0.12	-0.44	-0.81	-0.56	0.16	0

key	fmov	tmov	rmov	bfmov	lfmov	hit
1370	-0.07	-0.3	-0.34	-0.44	-0 05	1
1371	0.36	-0.12	-0.44	-0.38	0.53	0
1372	0.16	0.01	0.03	-1	0.22	1
1373	-0.06	0.16	0.34	-1.03	-0.07	0
1374	-5.12	0.37	0.25	-1.38	-4.99	0
1375	0.13	0.47	0.53	-0.63	0.14	1
1376	0.55	0.57	0.53	-0.75	0.66	1
1377	0.47	0.64	1.13	-0.91	0.56	1
1378	0.27	0.8	0.19	-1.94	0.33	1
1379	0.77	0.98	0.41	-1.03	0.82	1
1380	0.59	1.25	0.88	-2.19	0.58	1
1381	0.36	1.45	1.44	-2.09	0.48	1
1382	0.65	1.65	1.72	-2.84	0.8	1
1383	0.29	1.84	2.25	-2.09	0.22	1
1384	0.95	1.98	2.5	-1.25	0.62	1
1385	-0.15	2.12	2.56	-0.63	-0.77	0
1386	0.46	2.28	2	-1.59	0.56	1
1387	0.29	2.48	1.69	-2.69	0.01	1
1388	-1.31	2.67	1.75	-2.38	-1.13	0
1389	2	2.85	3.22	-2.09	1.38	1
1390	-0.22	2.99	2.53	-1.38	-0.73	0
1391	-1.07	3.14	4.31	-0.38	-1.26	0
1392	-0.51	3.19	3.88	0.69	-0.8	0
1393	-20.15	3.26	4.31	0.53	-20.87	0
1394	-0.45	3.32	3.88	-0.03	-0.52	0
1395	-1.3	3.34	3.16	-0.94	-1.77	0
1396	0.69	3.34	3.56	-0.81	0 41	1
1397	1.3	3.33	4.09	-0.09	0 64	1
1398	-4.31	3.24	5.06	2.75	-4.71	0
1399	-12.2	3.19	4.97	2.63	-13.07	0
1400	-2.32	3.15	5.31	2.16	-2.8	0
1401	-3.04	3.03	3.25	1.31	-3.47	0
1402	1.03	2.93	3.84	2	1.07	1
1403	-3.44	2.72	2.06	1.06	-3.83	0
1404	-32.78	2.54	2.97	1.66	-33.9	0
1405	0.73	2.37	3.09	1.38	0.95	1
1406	-0.38	2.2	2.78	1.75	-0.36	0
1407	0.32	2.04	2.53	2.16	0.48	1
1408	0.46	1.85	2.28	2.59	0.71	1
1409	-3.46	1.62	0.09	0.66	-3.57	0
1410	1.88	1.33	0.34	1.5	2.68	1
1411	0.98	1.03	1.03	2.38	1.55	1
1412	2.35	0.74	0.25	1.59	3.49	1
1413	0.25	0.48	0.34	1.06	0.52	1
1414	1.37	0.18	-0.5	0.88	2.91	0
1415	1.87	-0.07	-0.25	1.59	3.08	0
1416	-0.03	-0.34	0.38	1.63	-0.15	0
1417	0.66	-0.61	-0.06	1 38	1.35	0

key	fmov	tmov	rmov	bfmov	lfmov	hit
1418	0.55	-0.87	-0.44	1.13	1.87	0
1419	0.77	-1.17	-0.97	1.34	1.75	0
1420	0.33	-1.47	-1.03	1.78	1.44	0
1421	-0.52	-1.65	-1.75	1.22	0.87	1
1422	0.13	-1.89	-1.91	1.13	0.99	0
1423	0.69	-2.13	-1.38	0.94	0.87	0
1424	-0.13	-2.34	-2.75	0.84	0.52	1
1425	0.43	-2.53	-3.09	0.41	0.77	0
1426	-0.49	-2.62	-3.69	-0.28	0.39	1
1427	-0.73	-2.79	-3.16	1.09	0.03	1
1428	-0.57	-2.98	-3	1.09	0.03	1
1429	-0.21	-3.16	-3.22	1.25	0	1
1430	-0.42	-3.34	-4.53	0.34	0.05	1
1431	-0.49	-3.48	-4.5	-0.25	-0.18	1
1432	-1.12	-3.6	-4.09	-0.47	-0.51	1
1433	-0.89	-3.67	-5.09	-0.59	-0.48	1
1434	0.25	-3.74	-4.63	0.38	0.27	0
1435	-0.25	-3.89	-4.47	0.5	-0.08	1
1436	0.01	-3.97	-4.16	0.69	0.04	0
1437	-0.74	-4.02	-2.47	0.84	-0.54	1
1438	0.13	-4.05	-4.22	0	0.19	0
1439	-1.13	-4.1	-3.88	-0.66	-0.85	1
1440	-0.15	-4.19	-4.28	-0.84	-0.15	1
1441	-0.68	-4.25	-4.66	-1.13	-0.65	1
1442	-0.91	-4.23	-4.19	-0.81	-0.89	1
1443	-0.34	-4.2	-3.81	-0.81	-0.4	1
1444	-0.46	-4.16	-3.25	-1.31	-0.57	1
1445	-0.23	-4.07	-3.53	-1.84	-0.37	1
1446	-0.4	-4	-4.91	-1.91	-0.57	1
1447	0.07	-3.93	-4.53	-2.22	-0.26	0
1448	-0.33	-3.87	-4.19	-2	-0.78	1
1449	-0.03	-3.89	-4.5	-2.13	-0.29	1
1450	0.04	-3.84	-4.19	-1.69	-0.56	0
1451	0.38	-3.85	-5.16	-1.5	-0.15	0
1452	0.12	-3.83	-4.56	-1.5	-0.43	0
1453	0.43	-3.78	-4.06	-1.56	-0.36	0
1454	0.81	-3.75	-3.88	-1.34	0.35	0
1455	0.49	-3.71	-3.06	-1.09	0.21	0
1456	0.45	-3.71	-3	-0.81	0.1	0
1457	0.93	-3.71	-3	-0.66	0.57	0
1458	1.07	-3.66	-2.84	-0.47	0.59	0
1459	1.26	-3.6	-2.75	-0.59	0.86	0
1460	0.95	-3.56	-3.09	-0.56	0.64	0
1461	1.68	-3.52	-2.91	-0.03	0.98	0
1462	1.83	-3.43	-4.25	-2.03	1.47	0
1463	0 89	-3.28	-3.72	-1.72	1.65	0
1464	2.17	-3.16	-3.59	-1.84	1.63	0
1465	1.66	-3.04	-3.53	-1.91	1.36	0

key	fmov	tmov	rmov	bfmov	lfmov	hit
1466	2.23	-2.92	-2.91	-1.56	1.64	0
1467	2.62	-2.8	-3.19	-1.66	2.09	0
1468	1.47	-2.67	-3.41	-2.22	1.47	0
1469	2.19	-2.55	-3.97	-1.69	1.8	0
1470	2.92	-2.44	-2.94	-1.13	2.43	0
1471	1.27	-2.31	-3.44	-2.06	0.84	0
1472	4.05	-2.14	-3.47	-2.31	3.51	0
1473	1.44	-1.99	-2.22	-0.88	1.21	0
1474	3.15	-1.75	-1.56	-0.66	2.86	0
1475	2.01	-1.61	-1.88	0.94	1.86	0
1476	5.64	-1.44	-1.41	0.03	5.11	0
1477	2.56	-1.31	-0.94	0.22	2.21	0
1478	1.59	-1.21	-0.31	0.19	1.25	0
1479	2.49	-1.05	-0.06	0.22	2.31	0
1480	-2.01	-0.91	-0.25	0.06	-2.36	1
1481	2.24	-0.77	-0.44	-0.19	2.31	0
1482	2.05	-0.7	0.44	1.25	2.11	1
1483	0.77	-0.64	0.63	1.47	0.73	1
1484	0.1	-0.54	0.69	0.06	0.05	1
1485	-11.65	-0.52	1.22	0.31	-14.01	0
1486	-17.78	-0.57	-0.41	-0.88	-18.52	1
1487	0.1	-0.58	0.13	-0.28	0.09	1
1488	2.87	-0.61	-0.41	-0.97	3.12	0
1489	2.65	-0.65	-0.59	-1.72	2.79	0
1490	-2.82	-0.73	0.47	-1.03	-3.22	0
1491	2.14	-0.9	-0.22	-0.69	2.35	0
1492	1.93	-1.06	-0.78	-0.88	1.89	0
1493	2.5	-1.2	-1.25	-1.13	2.63	0
1494	-4.5	-1.42	-2.16	-1.31	-5.27	1
1495	4.4	-1.66	-1.06	0	4.92	0
1496	-0.56	-1.86	-1.88	-0.53	-0.77	1
1497	-18.09	-2.08	-2.56	-0.56	-20.04	1
1498	-11.57	-2.22	-2.22	-0.72	-12.53	1
1499	-8.25	-2.36	-2.06	-0.41	-8.99	1
1500	-3.59	-2.45	-1.84	-0.59	-3.91	1
1501	2.49	-2.54	-2.06	-0.66	2.85	0
1502	10.11	-2.69	-4.13	-1.09	6.96	0
1503	-0.7	-2.84	-3.97	-1	-0.96	1
1504	-11.55	-3	-3.5	-0.94	-12.42	1
1505	0.34	-3.15	-4.59	-1	0.07	0
1506	3.06	-3.26	-5	-1.06	2.9	0
1507	-0.83	-3.33	-3.97	-1.59	-1.31	1
1508	-34.95	-3.44	-3.75	-0.78	-37.09	1
1509	-1.57	-3.5	-3.75	-0.91	-2.19	1
1510	-13.54	-3.55	-3.09	-0.59	-14.54	1
1511	-0.35	-3.61	-2.47	0	-0.95	1
1512	1.94	-3.74	-2.56	0.44	1.68	0
1513	-9.98	-3.88	-2.91	-0.13	-10.86	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1514	0.6	-3.98	-3.75	0.22	0.26	0
1515	2.7	-4.09	-4.47	0.41	2.99	0
1516	-0.75	-4.23	-4.63	1.22	-1.02	1
1517	3.86	-4.32	-4.66	1.13	4.38	0
1518	-16.46	-4.4	-2.88	2.03	-17.8	1
1519	7.27	-4.52	-4.28	1.38	5.43	0
1520	-0.75	-4.66	-3.97	1.81	-0.94	1
1521	-1.84	-4.85	-3.31	1.78	-2.17	1
1522	2.71	-5.08	-3.47	2.03	3	0
1523	-4.85	-5.39	-4.91	2.13	-5.17	1
1524	-24.25	-5.71	-5.28	2.41	-24.81	1
1525	2.77	-6.02	-6.47	1.56	3.21	0
1526	-2.61	-6.26	-6.44	0.75	-2.45	1
1527	0.96	-6.48	-6.75	-0.09	3.3	0
1528	2.76	-6.79	-6.72	0.75	3.37	0
1529	-0.06	-7.04	-6.69	0.09	-0.2	1
1530	-6.96	-7.37	-6.84	-0.53	-7.59	1
1531	2.97	-7.59	-6.84	-0.44	3.94	0
1532	1.24	-7.81	-8.06	-1.31	1.84	0
1533	0.45	-8.03	-8.53	-1.69	0.9	0
1534	-0.43	-8.24	-9.53	-2.78	-0.39	1
1535	0.97	-8.46	-9.94	-3.03	1.36	0
1536	0.21	-8.64	-10.13	-2.63	0.4	0
1537	4.94	-8.75	-9.22	-2.63	5.43	0
1538	3.26	-8.83	-9.5	-2.97	3.61	0
1539	0.19	-8.87	-11.72	-3.56	0.18	0
1540	-0.48	-8.89	-10.56	-4.59	-0.65	1
1541	-4.36	-8.91	-10.25	-4.06	-4.96	1
1542	-7.04	-8.92	-9.5	-3.38	-7.95	1
1543	-5.77	-8.91	-8.97	-3.75	-6.68	1
1544	0.64	-8.8	-8.41	-3.94	0.76	0
1545	-0.71	-8.71	-8.38	-2.63	-1.84	1
1546	-7.86	-8.55	-9.75	-1.75	-8.85	1
1547	-10.23	-8.35	-9.47	-2.25	-11.34	1
1548	-6.52	-8.19	-9.09	-1.44	-7.8	1
1549	-7.26	-8.09	-8.28	-1.22	-8.51	1
1550	-6.27	-7.99	-7.59	-1.16	-8.73	1
1551	-0.78	-7.78	-7.19	0.13	-3.4	1
1552	-9.07	-7.64	-7.13	-0.78	-13.13	1
1553	-2.62	-7.49	-7.09	-1.91	-4.55	1
1554	-9.5	-7.4	-6.66	-1.03	-11.17	1
1555	1.38	-7.3	-5.63	-0.72	0.02	0
1556	-9.73	-7.31	-6.25	-0.5	-12.42	1
1557	-14.05	-7.3	-5.88	-1.31	-17.69	1
1558	-12.63	-7.2	-5.44	-1.13	-16.25	1
1559	-3.11	-7.21	-6.38	-0.81	-7.53	1
1560	2.99	-7.22	-6.97	-1	1.63	0
1561	-5.3	-7.26	-7.22	-1.91	-9.3	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1562	-9.68	-7.27	-6.88	-2.5	-13.23	1
1563	-11.41	-7.27	-7.25	-2.5	-14.29	1
1564	-3.54	-7.27	-7	-1.56	-5.11	1
1565	-6.16	-7.24	-7.31	-0.91	-8.58	1
1566	-13.58	-7.17	-6.78	-0.94	-17.06	1
1567	-3.84	-7.14	-8.53	-1.78	-6	1
1568	-9.2	-7.12	-8.06	-1.19	-13 17	1
1569	-6.59	-7.09	-7.5	-1.06	-8.69	1
1570	-0.71	-7	-9.66	-2.97	-1.47	1
1571	3.12	-6.84	-9.41	-3.03	2.64	0
1572	-14.4	-6.65	-9.25	-2.5	-18.25	1
1573	-8.01	-6.4	-7.72	-2.25	-10.45	1
1574	0.04	-6.09	-7.25	-2.09	-1.62	0
1575	-1.89	-5.76	-7.09	-2.03	-3.75	1
1576	-1.14	-5.41	-6.34	-2.84	-2.48	1
1577	-0.92	-4.97	-5.13	-4.25	-1.84	1
1578	6.99	-4.64	-4.94	-3.78	6.89	0
1579	-13.53	-4.23	-5.84	-3.94	-18.55	1
1580	1	-3.89	-5.16	-3.09	-0.11	0
1581	0.44	-3.54	-3.34	-2.22	-0.51	0
1582	-1.96	-3.13	-2.63	-1.56	-3.61	1
1583	-0.12	-2.67	-2.78	-1.63	-1.95	1
1584	2.9	-2.09	-1.31	-2.84	1.78	0
1585	-0.43	-1.59	0.13	-1.97	-0.83	0
1586	1.27	-1.15	0.34	-2.5	0.25	1
1587	1.82	-0.78	1.16	-1.75	0.09	1
1588	-0.41	-0.41	2.84	0.56	-1.24	0
1589	-0.9	-0.05	0.75	-1.22	-0.18	0
1590	-0.16	0.29	1	-0.94	-1.41	0
1591	-1.49	0.67	-0.34	-0.97	-3.41	1
1592	-0.82	1.07	0.5	-1.44	-2.7	0
1593	0.59	1.43	-0.28	-1.16	-1.28	0
1594	6.65	1.7	1.25	-0.5	5.65	1
1595	17.88	1.89	4.16	2.13	18.71	1
1596	-0.3	2.06	3.69	1.78	-1.02	0
1597	0.16	2.23	2.88	1.28	-0.11	1
1598	0.17	2.4	1.53	0.09	0.42	1
1599	0.27	2.59	2.06	-0.78	0.04	1
1600	-0.09	2.67	3.13	0.44	-0.47	0
1601	0.08	2.92	3.06	-0.91	0.37	1
1602	1.77	3.12	2.75	-1.81	1.93	1
1603	0.54	3.43	4.13	-1.69	0.79	1
1604	-0.77	3.67	4.91	-1.88	-1.57	0
1605	0	3.99	3.47	-2.78	0.64	0
1606	0.25	4.18	1.75	-3.5	0.94	1
1607	0.44	4.24	2.59	-3.19	0.87	1
1608	-0.05	4.33	3.91	-4.09	-0.92	0
1609	0.65	4.46	4.28	-3.44	0.92	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1610	0.41	4.67	5.59	-3.94	0 46	1
1611	-1.25	4.86	4.75	-3.34	-2.1	0
1612	1.63	5	6.38	-2.5	1 43	1
1613	0.49	5.12	5.63	-1.66	0 35	1
1614	0.35	5.25	6.75	0.06	-0.02	1
1615	2.44	5.3	6.09	0.72	0 69	1
1616	1.62	5.27	6.97	1.81	2 05	1
1617	0.21	5.32	5.78	1.03	1 04	1
1618	0.04	5.41	5.41	1.28	-0.1	1
1619	0.43	5.46	5.78	3.34	0 61	1
1620	-0.02	5.44	5.84	3.75	-0.06	0
1621	0.63	5.4	6.38	3.97	1 23	1
1622	2.68	5.26	6.53	4.16	4.54	1
1623	-0.01	5.12	6.31	5.28	0 68	0
1624	0.8	4.91	5.88	5.41	1 69	1
1625	1.14	4.72	5.59	4.66	2.26	1
1626	0.11	4.47	5.38	3.72	0.97	1
1627	-0.79	4.32	4.16	1.91	1.22	0
1628	0.32	4.08	4.66	2.91	0.83	1
1629	-0.02	3.85	3.81	1.72	0.01	0
1630	9.59	3.53	3.63	1.63	13.51	1
1631	-0.53	3.26	3.47	0.78	0.59	0
1632	-0.18	2.94	3.47	2.41	0 12	0
1633	-0.2	2.57	2.28	2.63	-0.23	0
1634	-0.03	2.19	1.56	1.59	1	0
1635	-0.38	1.83	1.69	0.03	-0.34	0
1636	9.68	1.54	1.25	-1.38	13.26	1
1637	-1.32	1.2	0.84	-0.22	-0.65	0
1638	-0.65	0.84	2.72	1.16	1.14	0
1639	-2.82	0.52	1.38	1.13	-1.69	0
1640	-0.52	0.15	0.59	2.13	-1.4	0
1641	-2.65	-0.22	-1.91	2.19	-2.14	1
1642	6.78	-0.65	-0.56	4.03	9.65	0
1643	-2.14	-0.97	-1.44	1.72	-2.04	1
1644	-2.24	-1.22	-2.19	-0.28	-1.92	1
1645	-4.29	-1.48	-2.13	0.69	-3.7	1
1646	-3.42	-1.73	-2.09	2.31	-2.54	1
1647	-0.39	-1.97	-0.72	2.84	-2.14	1
1648	-1.19	-2.18	-2.19	2.75	-0.05	1
1649	-0.78	-2.35	-2.97	2.06	-2.35	1
1650	-0.76	-2.58	-3.19	2.41	0 12	1
1651	2.81	-2.74	-3.94	1.06	1.98	0
1652	-2.82	-2.91	-4.63	0.94	-3.24	1
1653	-2.15	-2.97	-6.16	-0.69	-2.96	1
1654	-6.82	-3.08	-3.91	0.72	-6.68	1
1655	4.1	-3.07	-2.41	0.53	4.19	0
1656	-2.56	-3.05	-3.56	-0.38	-3.42	1
1657	-3.68	-2.97	-4.25	-2.22	-3.81	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1658	-5.32	-2.93	-3.81	-1.88	-5.57	1
1659	-3.39	-2.94	-3.53	-3.16	-3.08	1
1660	-3.38	-2.9	-3.16	-1.56	-3.53	1
1661	-4.17	-2.82	-2.53	-2.56	-4.68	1
1662	-3.71	-2.74	-2.38	-1.41	-4.1	1
1663	-2.57	-2.65	-3.25	-1.09	-3.66	1
1664	-3.54	-2.59	-3.38	0	-4.26	1
1665	-3.59	-2.42	-3.09	-1	-4.25	1
1666	-3.01	-2.36	-1.25	-0.09	-3.91	1
1667	-2.23	-2.36	-1.56	-0.31	-3.51	1
1668	-2.7	-2.29	-0.38	1.19	-3.74	1
1669	6.54	-2.18	-1.06	1.38	6.84	0
1670	-3.8	-2.08	-1	1.06	-4.68	1
1671	-2.53	-1.97	-1.22	-0.09	-3.36	1
1672	-2	-1.88	-1.19	0.53	-3.14	1
1673	-1.86	-1.73	-1.28	1.22	-2.84	1
1674	-2.78	-1.65	-2.06	0.31	-3.73	1
1675	-1.89	-1.52	-3.09	-0.81	-2.91	1
1676	-1.77	-1.39	-2.28	0.47	-3.23	1
1677	-1.69	-1.25	-2.59	-0.38	-3.07	1
1678	-1.41	-1.13	-2.31	-1.5	-2.42	1
1679	-1.12	-0.95	-2.06	-3.91	-2.49	1
1680	-1.44	-0.84	-1.63	-3.25	-2.63	1
1681	-1.23	-0.71	-1.59	-2.5	-2.29	1
1682	-1.7	-0.62	-1.06	-1.84	-2.86	1
1683	-0.72	-0.51	-1.03	-2.09	-2.16	1
1684	-0.43	-0.34	0.91	-3.75	-1.68	0
1685	-1	-0.19	-0.41	-3.34	-2.29	1
1686	-0.97	0.02	-0.34	-3.25	-2.16	1
1687	-0.7	0.3	-0.38	-4.53	-1.86	1
1688	1.54	0.55	0.22	-3.44	-0.21	1
1689	-0.34	0.82	1.44	-2.41	-1.77	0
1690	0.43	1.08	2.59	-0.53	-1.01	1
1691	0.57	1.37	2.16	-1.56	-0.76	1
1692	1.28	1.57	1.78	-0.75	-0.15	1
1693	2.17	1.73	1.22	-0.5	0.56	1
1694	1.25	1.91	1.19	-0.75	-0.01	1
1695	2.99	2.06	2.72	0.16	1.52	1
1696	1.36	2.12	2.19	-0.25	0.59	1
1697	1.23	2 17	2.94	1.41	0.23	1
1698	1.42	2.24	3.34	1.88	0.68	1
1699	1.41	2.27	3.44	3.06	0.71	1
1700	3.78	2.28	3.47	2.63	2.94	1
1701	1.67	2.25	3.84	2.31	1.08	1
1702	1.34	2.22	4.56	2.09	0.92	1
1703	2.1	2.2	2.81	1.53	1.47	1
1704	1.75	2.18	2.28	1.69	1.53	1
1705	1 67	2.22	2.94	1.63	1.38	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1706	-0.17	2.3	2.47	1.66	-0.19	0
1707	0.38	2.35	2.28	0.88	0.88	1
1708	0.36	2.47	0.78	-0.63	0.69	1
1709	0.23	2.54	1.16	0.38	0.74	1
1710	-0.51	2.57	0.31	-1.22	-0.55	0
1711	-0.35	2.59	0.59	-0.31	0.02	0
1712	0.18	2.64	0.66	0.09	0.67	1
1713	-0 64	2.73	1.84	0.28	-0.33	0
1714	-0.63	2.78	1.72	-0.19	-0.17	0
1715	-0.89	2.95	1.34	-1.44	-0.45	0
1716	-0.77	3.15	2.13	-1.06	-0.43	0
1717	-0.37	3.31	3.03	-0.5	-0.36	0
1718	-0.96	3.49	3.94	0.81	-0.59	0
1719	-0.59	3.68	4.97	0.47	0.07	0
1720	-0.79	3.96	4.44	0.06	-0.33	0
1721	-0.43	4.21	4.22	0.69	-0.53	0
1722	-0 88	4.38	3.81	3.41	-0.66	0
1723	-1.35	4.55	4.66	1.97	-1.22	0
1724	-1.1	4.76	5.81	1.94	-1.13	0
1725	-1.25	4.88	5.78	2.38	-1.16	0
1726	-1.98	5.02	6.75	2.44	-1.99	0
1727	-2.12	5.17	6.72	2.25	-2.11	0
1728	-1.49	5.29	6.66	0.75	-1.41	0
1729	-1.32	5.36	6.59	0.53	-1.27	0
1730	-1.74	5.39	6.66	0.88	-1.5	0
1731	-1 91	5.38	7.25	0.44	-1.94	0
1732	-1.21	5.38	7	0.13	-1.33	0
1733	-1.56	5.4	4.06	-1.94	-1.51	0
1734	-1.24	5.49	4.72	-0.75	-1.5	0
1735	-1.54	5.52	5.28	0.03	-1.49	0
1736	-1.46	5.53	4.69	-0.81	-1.66	0
1737	-1.29	5.51	5.06	-0.22	-1.68	0
1738	-2 43	5.47	4.63	-0.38	-2.85	0
1739	-2.28	5.44	5.03	0.72	-2.37	0
1740	-1.87	5.37	4.56	0.84	-1.64	0
1741	-1.6	5.32	4.72	0.28	-2.01	0
1742	-1.9	5.25	4.75	0.56	-1.86	0
1743	-2 42	5.18	4.41	1	-2.3	0
1744	-2	5.18	4.56	-0.09	-2.12	0
1745	-0.46	5.33	5.97	-0.16	-0.01	0
1746	-2.09	5.46	5.28	-0.75	-1.89	0
1747	-2.09	5.57	6	-0.31	-2.24	0
1748	-2.43	5.68	5.28	-0.06	-2.87	0
1749	-1.35	5.83	5.97	-0.25	-2.14	0
1750	0.83	5.97	5.97	-0.88	-0.55	1
1751	-1.19	6.09	5.09	-0.63	-2.15	0
1752	-1.35	6.19	5.31	0.44	-1.91	0
1753	-1.5	6.29	5.06	0.75	-1.94	0

key	fmov	tmov	rmov	bfmov	lfmov	hit
1754	-1.2	6.39	5.84	0.81	-1.63	0
1755	-1 56	6.47	6.78	2.22	-2.2	0
1756	-1.69	6.54	7.56	2.53	-1.77	0
1757	-2.3	6.49	7.72	3.56	-2.79	0
1758	-0.11	6.49	7.75	3.66	0.15	0
1759	-0.35	6.48	7.44	2.69	0.23	0
1760	0.09	6.51	8.31	2.91	0.79	1
1761	-0.66	6.46	7.91	4.09	0.01	0
1762	-0.56	6.4	7.72	4.25	0.05	0
1763	-0.75	6.4	7.03	3.88	0.11	0
1764	6.01	6.39	6.94	3.97	7.48	1
1765	-1.17	6.37	7.03	3.53	-0.15	0
1766	3.96	6.3	6.28	3.13	5.39	1
1767	2.02	6.15	6.13	3.47	4.28	1
1768	-2.35	5.97	4.84	2.16	-1	0
1769	4.48	5.74	5.34	2.88	6.41	1
1770	1.27	5.53	5.63	3.47	2.79	1
1771	-6.7	5.32	6.16	3.66	-5.63	0
1772	8.17	5.07	4.81	2.03	11.01	1
1773	4.32	4.83	4.56	1.41	6.28	1
1774	5.33	4.61	5.06	1.28	7.52	1
1775	-2.41	4.41	4.97	0.75	0.51	0
1776	0.52	4.23	4.66	0.16	3.25	1
1777	-0.44	4.08	4.28	-0.88	2.07	0
1778	-2.11	3.96	3.34	-1.5	-0.79	0
1779	-1.65	3.88	3.34	-1.06	-0.12	0
1780	11.93	3.86	2.53	-2.06	14.16	1
1781	-2.46	3.82	2.94	-2.47	-1.07	0
1782	-2.38	3.69	2.53	-1.44	-1.33	0
1783	-2.98	3.47	2.63	0.5	-2.77	0
1784	-3.22	3.33	2.41	0.03	-2.82	0
1785	-3.28	3.17	2.5	0.5	-2.86	0
1786	-1.27	3	2.53	0	-0.77	0
1787	-3.12	2.81	2.72	0.97	-3.16	0
1788	-2.11	2.68	3.56	1.66	-1.64	0
1789	-2.35	2.59	3.56	1.38	-2.53	0
1790	-5.26	2.53	4.41	0.88	-5.4	0
1791	-2.37	2.46	4.22	0.88	-2.82	0
1792	-1.7	2.41	4.56	2.44	-1.5	0
1793	-2.34	2.32	2.66	2.13	-2.72	0
1794	-3.11	2.25	1.09	1.44	-3.61	0
1795	-1.99	2.17	1.5	1.69	-2.41	0
1796	-2.58	2.13	0.84	1.44	-2.71	0
1797	-4 3	2.11	1 25	2.09	-4.79	0
1798	-3.63	2.04	0 63	2.09	-4.51	0
1799	-3.82	2	1 53	1.06	-4.6	0
1800	7.2	1.89	2.22	2	7.96	1
1801	-0.32	1.79	1 94	2.16	0.24	0

key	fmov	tmov	rmov	bfmov	lfmov	hit
1802	3.92	1.66	1.72	2.63	7.26	1
1803	-1.26	1.48	1.41	2	-0.79	0
1804	-1.54	1.25	1	1.72	-0.65	0
1805	15.07	1.12	0.81	1.09	17.17	1
1806	-7.77	1.07	0.84	0.69	-6.93	0
1807	-7.61	0.96	1.44	1.25	-7.42	0
1808	-3.06	0.89	2	0.88	-2.24	0
1809	-3.63	0.79	0.91	0.75	-3.55	0
1810	16.07	0.75	1.88	1	19.23	1
1811	-1.98	0.65	1.09	1.09	-3.05	0
1812	-6.87	0.51	1.19	1.91	-6.73	0
1813	2.65	0.33	1.47	2	4.2	1
1814	0.77	0.18	0.06	1.16	2.17	1
1815	-7.63	0.02	-0.63	1.13	-7.72	1
1816	-0.96	-0.08	-0.44	0.69	-1.33	1
1817	-1.4	-0.15	0.06	0.66	-1.9	0
1818	-1.12	-0.25	-1.09	0.06	-2.11	1
1819	-7.73	-0.37	-0.84	0.78	-8.03	1
1820	-6.87	-0.49	-0.94	0.03	-7.45	1
1821	-5.36	-0.53	-0.44	-0.16	-5.82	1
1822	-1.93	-0.65	-0.59	-0.75	-2.08	1
1823	-3.02	-0.73	-1.09	-1.28	-3.89	1
1824	-6.77	-0.84	-2.13	-1.69	-7.84	1
1825	-1.21	-0.95	-1.84	-1.78	-2.16	1
1826	-3.52	-1.01	-2.31	-1.59	-4.12	1
1827	-4.81	-1.04	-1.16	-1.28	-5.72	1
1828	0	-1.12	-0.94	0	-0.62	1
1829	-0.69	-1.18	-1.34	-0.38	-1.07	1
1830	-0.33	-1.2	-1.28	-0.84	-0.92	1
1831	-4.81	-1.19	-0.75	-1.31	-5.67	1
1832	-2.21	-1.17	-0.22	-1.03	-3.55	1
1833	-0.35	-1.18	-0.69	-0.88	-0.72	1
1834	-1.83	-1.16	-0.94	-1.16	-2.93	1
1835	-0.07	-1.12	-1.34	-1 28	-0.52	1
1836	-0.79	-1.03	-0.88	-1 13	-2.28	1
1837	-0.17	-1.02	-1.41	-0.38	-1.02	1
1838	0.52	-1.02	-1.44	0.03	-0.35	0
1839	-7.23	-1.03	-2.13	-1 59	-9.16	1
1840	0.7	-1.04	-1.34	-1.63	-0.01	0
1841	0.95	-1.05	-1.5	-0.72	-0.41	0
1842	0.05	-1.07	-0.63	0.47	-0.9	0
1843	-0.21	-1.14	-0.47	1.16	-0.89	1
1844	0.08	-1.24	-0.66	1.03	0.2	0
1845	0.48	-1.33	-0.31	2.03	-0.1	0
1846	0.48	-1.46	0	2.84	0.17	1
1847	0.13	-1.55	-0.06	2.38	0.22	0
1848	0.33	-1.66	-1.66	1 41	0.22	0
1849	-0.84	-1.73	-2.34	0.88	-0.78	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1850	-0.2	-1.79	-1.41	1.41	-0.52	1
1851	-0.17	-1.85	-1.06	2.38	-0.07	1
1852	-0.28	-1 91	-1.75	0.66	-0.21	1
1853	-1.1	-1.96	-1.59	0.09	-1.22	1
1854	-0.2	-2.05	-2.31	-0.16	-0.22	1
1855	-0.59	-2.15	-2.56	-0.09	-0.81	1
1856	-0.08	-2.18	-2.75	-1.28	-0.4	1
1857	-0.11	-2.23	-4.03	-1.97	-0.23	1
1858	-0.68	-2.28	-3.34	-1.09	-1.02	1
1859	0.94	-2.3	-3.47	-1.31	0.65	0
1860	-0.2	-2.27	-2.97	-0.59	-0.43	1
1861	-0.09	-2.19	-2.84	-0.38	-0.36	1
1862	-0.54	-2.13	-3.53	-1.59	-0.68	1
1863	-0.98	-2.06	-2.72	-0.75	-1.17	1
1864	-0.5	-1.97	-2.66	-0 56	-0.7	1
1865	-0.07	-1.9	-2.75	-0.03	-0.41	1
1866	-0.47	-1.8	-2.69	-0.34	-0.45	1
1867	-1.19	-1.67	-1.22	-0.19	-1.43	1
1868	0.06	-1.55	-1.47	0.34	-0.29	0
1869	-0.28	-1.37	-1.22	0.22	-0.69	1
1870	0.03	-1.21	-0.56	0.06	-0.04	0
1871	0.17	-1.04	-1.03	-0.84	-0.3	0
1872	-0.06	-0.87	-0.5	-1.13	-0.29	1
1873	0.14	-0.71	0.06	-0.91	-0.08	1
1874	0.35	-0.52	0.44	-0.81	0.05	1
1875	0.48	-0.36	0.34	-0.69	-0.09	1
1876	0.17	-0.18	0.16	-1.56	0.26	1
1877	0.44	0	-0.09	-1.19	0.09	0
1878	0.25	0.16	0.53	-0.38	-0.31	1
1879	0.62	0.28	-0.09	-1.06	0.46	0
1880	0.31	0.41	0.06	-1.19	0.4	1
1881	0.63	0.58	0.44	-1.28	0.42	1
1882	0.49	0.73	0.31	-0.84	0.46	1
1883	1.1	0.9	0 94	-0.53	0.97	1
1884	0.17	1.03	0.91	0.06	-0.26	1
1885	0.61	1.15	0.94	-0.31	0.59	1
1886	0.57	1.27	0.88	-0.63	0 66	1
1887	0.74	1.4	1.38	-0.88	0.68	1
1888	0.56	1.54	1.53	-0.88	0.3	1
1889	0.49	1.71	1.03	-1.34	0.81	1
1890	0.47	1.83	1.44	-0.97	0.5	1
1891	0.94	1.93	1.56	-0.5	0.81	1
1892	1.08	2.06	2.78	-0.69	1.16	1
1893	1.62	2.19	2.75	-0.88	1.34	1
1894	1.15	2.31	2.91	-0.41	1.15	1
1895	0.83	2.45	2.53	-1.19	0.39	1
1896	1.17	2.58	2.91	-0.91	1.38	1
1897	1.56	2.69	3.09	-0.41	1 56	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1898	0.54	2.82	3.22	0.25	1.21	1
1899	1.01	2.95	3.47	-0.03	0.94	1
1900	0.84	3.08	3.72	-0.09	1.37	1
1901	1.07	3.26	3.34	-0.31	0.83	1
1902	1.62	3.44	2.25	-0.97	1.67	1
1903	1.4	3.59	3.13	-0.59	1.33	1
1904	2.6	3.69	3.38	-0.22	2.93	1
1905	1.47	3.78	3.16	0.25	1.29	1
1906	1.29	3.87	4.03	0.59	1.66	1
1907	0.86	3.96	3.84	1.03	1.54	1
1908	0.86	4.01	3.5	1.59	0.79	1
1909	0.62	4.09	4	1.34	0.48	1
1910	1.83	4.14	4.38	1.97	2.15	1
1911	1.41	4.18	4.44	2	1.89	1
1912	1.42	4.2	5.25	2.41	1.83	1
1913	1.04	4.26	5.41	2.44	1.74	1
1914	0.82	4.37	5.03	2.88	1.49	1
1915	0.75	4.43	5.19	2.25	1.49	1
1916	0.03	4.47	4.84	1.56	0.81	1
1917	-0.04	4.53	4.88	1.72	0.83	0
1918	0.01	4.53	4.75	1.5	0.79	1
1919	-0.11	4.5	3.94	1.06	0.61	0
1920	-0.54	4.54	4.88	0.5	0.35	0
1921	-0.59	4.56	4.47	0.31	0.24	0
1922	-1.61	4.59	4.28	-0.06	-0.54	0
1923	-0.89	4.6	4.25	-0.31	-0.08	0
1924	0.03	4.57	4.84	-0.16	0.27	1
1925	-0.94	4 51	4.75	-0.34	-0.33	0
1926	-1.1	4.47	4.53	0.44	-0.5	0
1927	-0.92	4.38	4.28	0.13	-0.3	0
1928	-1.21	4.26	4.38	0.47	-0.63	0
1929	-1.02	4.16	4	0.31	-0.54	0
1930	-0.78	4.03	3.34	1.06	-0.59	0
1931	-1.51	3 93	4 41	1.59	-1.11	0
1932	-1.08	3.78	4 44	1.44	-0.73	0
1933	-1.2	3.65	5.06	1.69	-0.99	0
1934	-0.61	3.5	4.63	1.66	-0.69	0
1935	-0.9	3.37	4 63	1.59	-0.98	0
1936	-0.23	3.2	4.06	0.97	-0.71	0
1937	-0.81	3.03	4.03	1.06	-1.01	0
1938	-1.63	2 87	3.03	1.34	-1.73	0
1939	-1.04	2.72	2.22	0.94	-1.29	0
1940	-0.09	2.62	2.56	0.19	-0.71	0
1941	-0.7	2.52	1.66	-0.34	-1.02	0
1942	-1.98	2.49	1.63	-0.25	-2.18	0
1943	-0.58	2.38	1.41	-0.16	-0.95	0
1944	-0.61	2.25	1.47	0.31	-1.07	0
1945	0.15	2.06	0.97	0.03	-0.2	1

key	fmov	tmov	rmov	bfmov	lfmov	hit
1946	-0.79	1.92	1.22	0	-1.15	0
1947	-1.48	1.74	1	1.03	-1.86	0
1948	-0.56	1.61	0.81	0.88	-0.92	0
1949	-1.3	1.47	0.81	0.63	-1.67	0
1950	-1.02	1.37	0.88	1.59	-1.47	0
1951	0.6	1.3	2	2.19	0.48	1
1952	-1.08	1.24	1.81	1.31	-1.81	0
1953	-0.7	1.2	2.47	1.59	-1.25	0
1954	-0.34	1.14	2	1.94	-1.02	0
1955	-1.51	1.1	1.31	1.44	-2.03	0
1956	-1.06	1.02	0.84	1.94	-1.95	0
1957	-0.95	0.97	1.38	2.03	-1.83	0
1958	-1.37	0.87	0.44	2.22	-1.96	0
1959	-2.75	0.74	1.06	2.06	-3.38	0
1960	0.1	0.6	0.81	1.84	-1.58	1
1961	-0.79	0.44	0.69	1.41	-1.76	0
1962	-4.77	0.26	0.69	1.88	-5.33	0
1963	-2.7	0.02	1.09	2.59	-3.1	0
1964	-1.24	-0.15	0 88	1.59	-2.82	0
1965	-3.16	-0.35	0.25	1	-3.26	0
1966	-4.68	-0.53	0.47	1.22	-4.96	0
1967	-2.74	-0.67	-0.44	0.75	-2.9	1
1968	-3.26	-0.75	-0.09	0.47	-4	1
1969	-4.46	-0.83	-1.06	0.03	-5.43	1
1970	-3.7	-0.9	-2.03	0.28	-4.23	1
1971	-6.02	-0.99	-2.5	0.72	-6.7	1
1972	0	-1.04	-2.88	0.06	0.67	0
1973	-6.5	-1.09	-3.25	-0.38	-7.73	1
1974	-6.19	-1.09	-3.44	-1.03	-8.06	1
1975	-8.75	-1.11	-2.22	-0.06	-10.83	1
1976	-3.51	-1.08	-1.97	-1.13	-4.05	1
1977	-4.11	-0.99	-2.31	-1 69	-5.36	1
1978	-5.25	-0.94	-1.81	-1.13	-6 58	1
1979	-3.74	-0.88	-0.91	-0.63	-5.96	1
1980	-2.18	-0.83	-0.63	-1.19	-2.61	1
1981	-11.04	-0.75	-1.13	-1.59	-14.1	1
1982	-2.26	-0.63	-1.03	-1.91	-3.07	1
1983	-3.95	-0.46	-0.38	-2	-5.34	1
1984	-1.28	-0.3	-0.31	-1.81	-1.77	1
1985	-7.63	-0.09	0.59	-2.09	-9.39	0
1986	-3.72	0.18	0.63	-2.16	-5.12	0
1987	-4.38	0.4	1.59	-0.88	-6.42	0
1988	-3.3	0.61	2.22	-0.41	-4 62	0
1989	-4.31	0.83	1.66	-0.38	-6 01	0
1990	-0.56	1.01	1.06	-0.19	-2.17	0
1991	-2.77	1.14	1.06	0.59	-4.21	0
1992	-3.92	1.23	0.66	0.44	-5.39	0
1993	-0.87	1.34	0.91	0.16	-1 68	0

key	fmov	tmov	rmov	bfmov	lfmov	hit
1994	-1.71	1.51	1.34	-0.09	-3.52	0
1995	-1.85	1.63	0.69	0.16	-2.99	0
1996	-1.03	1.8	1.69	0.47	-1.84	0
1997	-2.12	1.87	2.84	1.13	-3.34	0
1998	-2.04	1.97	2.63	0.63	-3.22	0
1999	-2.38	1.98	2.91	1.41	-3.3	0
2000	-0.13	1.97	2 91	0.88	-0.68	0